目　　录

第 1 章　图像科学综述

近几年来，图像处理与识别技术得到了迅速的发展。现在人们已充分认识到图像处理和识别技术是认识世界、改造世界的重要手段。目前它已应用于许多领域，成为 21 世纪信息时代的一门重要的高新科学技术。本章通过介绍图像处理与识别技术的基本概念、基本原理、数学表示模型、基本处理方法、发展过程、应用领域及展望，使读者对图像科学有一个整体的了解，为后续章节介绍的知识提供详细的背景知识。

1.1　引　言

1.1.1　图像处理与识别技术概述

图像就是用各种观测系统以不同形式和手段观测客观世界而获得的，可以直接或间接作用于人眼而产生视知觉的实体。科学研究和统计表明，人类从外界获得的信息约有 75%来自于视觉系统，也就是说，人类的大部分信息都是从图像中获得的。

图像处理是人类视觉延伸的重要手段，可以使人们看到任意波长上所测得的图像。例如，借助伽马相机、X 光机，人们可以看到红外和超声图像；借助 CT 可看到物体内部的断层图像；借助相应工具可看到立体图像和剖视图像。1964 年，美国在太空探索中拍回了大量月球照片，但是由于种种环境因素的影响，这些照片是非常不清晰的，为此，美国喷射推进实验室（JPL）使用计算机对图像进行处理，使照片中的重要信息得以清晰再现。这是这门技术发展的重要里程碑。此后，图像处理技术在空间研究方面得到广泛的应用。

总体来说，图像处理技术的发展大致经历了初创期、发展期、普及期和实用化期 4 个阶段。初创期开始于 20 世纪 60 年代，当时的图像采用像素型光栅进行扫描显示，大多采用中、大型机对其进行处理。在这一时期，由于图像存储成本高，处理设备造价高，因而其应用面很窄。20 世纪 70 年代进入了发展期，开始大量采用中、小型机进行处理，图像处理也逐渐改用光栅扫描显示方式，特别是出现了 CT 和卫星遥感图像，对图像处理技术的发展起到了很好的促进作用。到了 20 世纪 80 年代，图像处理技术进入普及期，此时的微机已经能够担当起图形图像处理的任务。VLSI 的出现更使得处理速度大大提高，其造价也进一步降低，极大地促进了图形图像系统的普及和应用。20 世纪 90 年代是图像技术的实用化时期，图像处理的信息量巨大，对处理速度的要求极高。

图像识别所讨论的问题，是研究用计算机代替人工自动地处理大量的物理信息，解决人类生理器官所不能识别的问题，从而部分代替人的脑力劳动。人类识别图像的过程总是先找出它们外形或颜色的某些特征进行比较分析、判断，然后加以分门别类，即识别它们。人们在研制自动识别机时也往往借鉴人的思维活动，采用同样的处理方法，然而图像的灰度与色彩是由光

强和波长不同的光波所引起，它们与景物表面的特性、方向、光线条件以及干扰等多种因素有关。在各种恶劣的工作环境里，图像与景物已有较大的差别。因此要区分图像属于哪一类，往往要经过预处理、分割、特征抽取、分析、分类、识别等一系列过程。现在这些技术完全可通过计算机进行模拟、对图像信息进行处理来达到对它的识别。

21 世纪的图像技术要向高质量化方面发展，主要体现在以下几点：①高分辨率、高速度，图像处理技术发展的最终目标是要实现图像的实时处理，这在移动目标的生成、识别和跟踪上有着重要意义；②立体化，立体化所包括的信息最为完整和丰富，数字全息技术将有利于达到这个目的；③智能化，其目的是实现图像的智能生成、处理、识别和理解。

1.1.2 图像处理与识别技术的应用领域

目前，图像处理与识别技术的主要应用领域有生物医学、文件处理、工业检测、机器人视觉、货物检测、邮政编码、金融、公安、银行、机械、交通、电子商务和多媒体网络通信等领域。

1. 遥感技术

遥感技术可以是飞机遥感和卫星遥感技术，过去，采用侦察飞机对地球上某些地区进行空中摄影后得来的照片需要较多的人力来判断分析，而现在改用配备有高级计算机的图像处理系统来判读分析，既节省人力，又加快速度，还可以从照片中提取人工所不能发现的大量的有用信息。从遥感卫星所获得的地球资源图片由于各种原因，图像质量欠佳，如果仍采用简单的直观判读的方法来分析这些图像显然是不合理，同时也是不能满足要求的，因此必须采用图像处理技术。如 LANDSAT 系列陆地卫星，采用多波段扫描器（MSS），在 900km 高空，对地球每一个地区以 18 天为一周期进行扫描成像，其图像分辨率约为地面上十几米或一百米。这些图像无论在成像、存储、传输过程中，还是在判读分析中，都必须采用很多的数字图像处理方法。目前遥感技术，尤其是卫星遥感，已经在资源调查、灾害监测、农业规划、城市规划、环境保护和军事等方面取得了重大的成果。

2. 医学图像处理

图像处理与识别技术在基础科学和临床中，都有极多的应用领域。例如对生物医学的显微图像处理分析，如红白细胞和细菌、染色体分析、胸部 X 线照片的鉴别、以及超声波图像的分析等都是医疗辅助诊断的有利工具，目前这类应用已经发展到专用的软件设备和硬件设备，最普遍的就是 CT 技术。近年来又出现了癌细胞涂片的识别辅助技术和最新的计算机辅助手术技术等。

3. 文字识别

文字识别应用目前非常广泛，涉及以下几个主要方面：交通管理中的汽车牌照识别、电子图书和办公自动化中的文档识别、多媒体视频图像中的注解文字识别、金融领域的银行支票的文字识别、图书封面的文字信息提取以及旅游业中服务于外国旅客的商标、交通路标的自动翻译等。

4. 工业领域中的应用

工业领域中的应用一般有以下几方面：机械加工零件表面和外观质量的自动检查和识别，装配和生产线零件尺寸自动化测量，弹性力学照片的应力分析和受力后的变形量分析。其中目

前发展比较热门的是“计算机视觉”，采用摄影和输入二维图像的机器人，可以确定物体的位置、方向属性以及其他状态，它可以完成普通的材料搬运，如AGV自导引小车的路径识别，部件装配，生产过程的自动监控，还可以在人不宜进入的环境里进行喷漆、焊接以及宇航探测等，如登月机器人和火星探测机器人。

1.1.3 图像处理与识别的技术内容

图像处理就是将图像转换成一个数字矩阵存放在计算机中，并采用一定的算法对其进行处理。图像处理的基础是数学，最主要的任务就是各种算法的设计和实现。目前的图像处理技术已经在许多不同的应用领域中得到重视，并取得了巨大的成就。根据应用领域的不同要求，可以将图像处理技术划分为许多分支，其中比较重要的分支如下。

（1）图像数字化：通过采样与量化过程将模拟图像变换成便于计算机处理的数字形式。图像在计算机内通常用一个数字矩阵来表示，矩阵中的每一个元素称为像素。图像数字化的设备主要是各种扫描仪与数字化仪。

（2）图像增强：主要目的是增强图像中的有用信息，削弱干扰和噪声，使图像清晰或将其转换为更适合人或计算机分析的形式。图像增强并不要求真实地反映原始图像。

（3）图像分割与特征提取：图像分割是将图像划分为一些互不重叠的区域，通常用来将分割的对象从背景中分离出来。图像的特征提取包括了形状特征、纹理特征和颜色特征等。

（4）图像分析：对图像中的不同对象进行分割、分类、识别、描述和解释。

上述图像处理的内容往往是相互联系的，一个实用的图像处理系统往往需要结合应用几种图像处理技术才能得到所需要的结果。例如，图像数字化是将一个图像变换为适合计算机处理的形式，这是图像处理的第一步；图像增强可以为后续进行的图像处理工作做准备，也可以是图像处理的最后目的；通过图像分割得到的图像特征既可以作为最后结果，也可以作为下一步图像分析和识别的基础。

图像处理技术涉及到的知识面很广，也很复杂。例如，进行图像识别需要掌握算法编程、随机过程和信号处理方面的知识，不少课题还需要更加专业的知识，如积分变换、神经网络、数学形态学等。另外，图像处理是一门应用性很强的学问，必须与计算机技术的发展相适应。例如，傅立叶变换是图像处理常用的方法。图像处理的另一个特点，也是难点，就是其算法的优劣与被处理对象的内容高度相关，很难找到一种适用于各种情况的通用方法。因此，图像处理按照处理的对象类别又可以分为遥感图像处理、医学图像处理等。

（5）模式识别：图像处理的重要目的之一就是识别，而模式识别技术也是图像技术重要性的体现，诸如指纹鉴别、人脸识别以及文字识别等，都要和模式识别打交道。

（6）人工智能：可以说图像处理、模式识别和人工智能是三位一体的学科。目前模式识别技术遇到的最大困难就是自动化程度不够，即智能程度不高。例如，人眼可以很容易从人群中找到自己要寻找的目标，但对于计算机来说，就连判断图像中是否有人存在这样看似简单的问题都很难解决。神经网络技术给人工智能开辟了一个新的方向，但目前该技术还不够成熟。

（7）计算机视觉：研究计算机视觉的目的是开发出能够理解自然景物的系统。在机器人领域中，计算机视觉能够为机器人提供眼睛的功能。

1.2 图像处理系统的基本构成

图像处理系统是执行图像处理、分析理解图像信息任务的计算机系统，虽然图像处理技术应用广泛，图像处理系统种类繁多，但它们的基本组成是相近的，主要包括图像输入设备、执行处理分析与控制的计算机、输出设备、及存储设备中的图像数据库、图像处理程序库与模型库。系统的结构原理框图如图 1-1 所示。

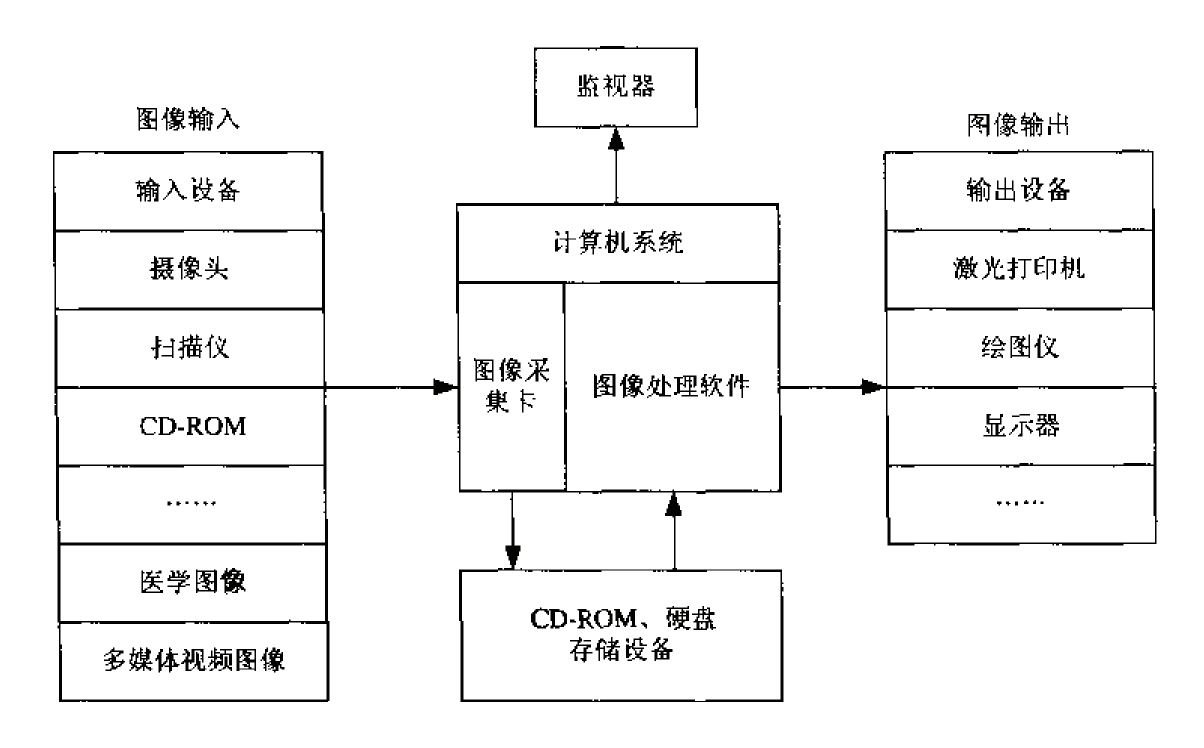


图 1-1 计算机图像处理系统示意图

数字图像处理与其他数据处理的不同之处是其庞大的数据处理量和存储量，以及对图像的显示。一帧 512×512 像素的真彩色图像，在不进行压缩的情况下，需要 780kB 的存储量和颜色数为 224 种的真彩色显示。因此，无论从硬件的配置还是软件环境上讲，计算机图像处理系统都有别于其他的计算机系统，从而形成了专门的图像处理计算机系统。

1.2.1 计算机图像处理系统的分类

计算机图像处理技术是以计算机为核心的应用技术，因此，计算机图像处理系统的发展，是随着计算机技术的提高而发展起来的。从系统的层次看，可分为高、中、低 3 个档次；从图像传感器的敏感区域看，又可分成可见光、红外、近红外、X 射线、雷达、伽玛射线、超声波等图像处理系统；从采集部件与景物的距离上来说，还可分成遥感、宏观和微观图像处理系统；就应用场所而言，又能分成通用图像处理系统和专用图像处理系统。通用系统一般用于研究开发，因此，要求传感器敏感区间宽，线性度好；而专用系统一般用于特殊用途，是在通用系统研究基础上，研制开发的为实现某一个或几个功能的商用系统。因此，在保证性能的前提下，由价格因素决定系统的配置。

（1）高档图像处理系统采用高速芯片设计，完全适合图像和信号处理特有规律的并行阵列图像处理机。这类系统采用多 CPU 或多机结构，可以以并行或流水线方式工作。

（2）中档图像处理工作站以小型机或工作站为主控计算机，加上图像处理器构成。这类系统有较强的交互处理能力，同时，由于用通用机做主控机，因而在系统环境下，具有较好的再开发能力。

（3）低档的微机图像处理系统由微机加上图像采集卡构成，其结构简单，是一种便于普及和推广的图像处理系统，也是本书着重介绍的系统。

1.2.2 微机图像处理系统的基本构成

微机图像处理系统由图像的采集部件、主机和图像的输出部件 3 部分组成。

1. 采集部件

原始的图像数据是通过图像采集部件进入计算机的，因此，图像采集部件的作用是采集原始的模拟图像数据，并将模拟信号转换成数字信号。计算机在接收到图像的数字信号后，将其存入内存。微机图像处理系统常用的图像采集部件有摄像机加上视频图像采集卡、图像扫描仪以及数码摄像机等。

（1）摄像机和视频图像采集卡

图像处理系统采用的摄像机分为电子管式摄像机和固体器件摄像机两种。电子管式摄像机根据光图像转换成电子图像的原理不同，可以分成光电子发射效应式和光导效应式两种类型，由于电子管式摄像机的重要元件是电子管，所以体积相对固体器件摄像机要大得多。目前普遍采用的固体器件摄像机是 CCD 类型的，CCD 摄像机是由电荷耦合元件组成的图像探测器，它将景物通过物镜成像在一块电荷感应光板（电荷耦合探测器）上，用感应光板上的感应电压模拟景物的亮度变化。当景物各点的光强度全部落在光电耦合器的线性感应区时，感应电压正比于景物各点的亮度变化，这时感应信号的失真度最小，如果景物亮度过亮或过暗，虽然人眼能够分辨出景物的特征，但是，图像数据的分布会出现极限饱和的情况，影响图像处理结果的正确性。由于用 CCD 实现了光电转换及扫描，因此其体积小、重量轻，结构紧凑。摄像机的参数有空间分辨率、灰度分辨率或颜色数、快门参数、最低照明度等。根据传感器的有效工作范围，可分为可见光、近红外、红外、X 射线等 CCD 摄像机；根据快门速度，可分为静止和实时摄像机。摄像机需要和视频图像采集卡配合使用，配合时，要考虑两者参数的优化问题。

视频图像采集卡可以将摄像机摄取的模拟图像信号转换成数字图像信号，使计算机得到所需要的数字化图像信号。一般视频图像采集卡都带有自己的帧缓冲存储器，用于存放所采集的图像数据。根据图像采集的速度，视频图像采集卡可以分为：中速采集卡、实时单帧采集卡和实时采集卡 3 种。中速采集卡采集速度大约是 1 帧/s，只能用于获得相对静止的图像，能满足一般科学研究的要求；实时单帧采集卡的采集速度为 40 帧/s，瞬时可采集 1 帧图像，因此可以用于获取任何活动的目标；实时采集卡的采集速度也为 40 帧/s，并能连续采集多帧图像，可以用于获得任何活动目标的运动过程，一般用于流水线上物品定时采样。视频采集卡插在微机的扩展槽上，并和摄像机连接使用。

（2）图像扫描仪

图像扫描仪也是一种获取图像，并将其转换成计算机可以显示、编辑、存储和输出的数字格式的设备，是一类适合于薄片介质，如纸张、照片（胶片）、插画、图形、树叶、硬币、纺织品等物体的图像数字化的设备。其空间分辨率较高，一般在 1200DH（点/英寸）以上。根据灰度分辨率的不同，可以分成黑白 64 级灰度扫描仪、黑白 256 级灰度扫描仪和彩色图像扫描

仪；根据面幅大小的不同，可以分成手提式扫描仪、平板式扫描仪、滚筒式扫描仪等；根据扫描仪结构的不同，可以分为透射式和反射式扫描仪。对于纸质图像的数字化，用扫描仪可以得到比摄像机和视频采集卡更为逼真的图像，但是，由于扫描仪以机械扫描的方式采集数据，因此采集速度不如 CCD 摄像机快。

（3）数码摄像机

数码摄像机是近年来出现的数字化产品，将图像采集和数字化部件集成在同一机器上，使其输出的信号能直接为计算机所接受。数码摄像机使图像的采集部件和主机的连接更具有通配性，而且由于其携带方便，有相应的存储器，因此更适用于现场数据采集。

2. 图像处理部件

在微机图像处理系统中，图像处理工作是由微机完成的，微机的扩展槽上插有带帧存储器的采集卡，图像处理的过程通常包含从帧存储器读取数据到计算机内存、处理内存中的图像数据和送数据回图像帧存储器 3 个步骤。对于直接使用内存的采集卡，则只需和内存进行数据交换，计算机的内存越大，CPU 的运算速度越快，图像处理的速度也越快。

3. 识别结果的输出部件

图像的输出是图像处理的最终目的。从广义的角度讲，图像的输出形式可以分为两种：

一种是根据图像处理的结果做出判断，例如质量检测中的合格与不合格，输出不一定以图像作为最终形式，而只需做出提示供人或机器做出选择。这种提示可以是计算机屏幕信息，或是电平信号的高低，这样的输出往往用于成熟研究的应用上。

另一种则是以图像为输出形式，它包括中间过程的监视以及结果图像的输出。图像输出方式有屏幕输出、打印输出和视频硬拷贝输出。

（1）屏幕输出

用屏幕输出处理结果是最直观、简单的方法，并可获得高质量图像。根据硬件的不同，可分为单屏显示和分屏显示两种形式。分屏显示是指图像处理的结果或中间过程由专门的监视器显示，加上计算机本身的显示器，这样的系统可以称作为双屏系统，由于图像部分和程序执行过程提示互不干扰，因此处理过程比较直观。单屏显示是指图像处理的过程与结果都在计算机的显示器上显示，一屏两用，比较经济。

（2）打印输出

打印输出的设备为打印机，按打印效果分成黑白、彩色两种，且有点阵式、喷墨式、激光式、热敏式打印机，黑白打印机只能打印黑白两色，灰度像素采用半色调的方式打印输出，即灰度差别用不同黑点数的表示方法。目前的彩色打印输出已有彩色喷墨打印机、彩色激光打印机、彩色热升华打印机等。

（3）视频硬拷贝输出

视频硬拷贝输出采用专用的拷贝和拷贝纸，得到高质量输出图像。视频拷贝机分成模拟式和数字式两种。模拟式拷贝机需连接视频信号，数字式拷贝机则可以直接和计算机相连，视频硬拷贝输出形式能长时间地保存图像。

1.3 图像的数字化与表示

图像能够以各种各样的形式出现，例如，可视的和不可视的、抽象的和实际的、适于计算

机处理的和不适于计算机处理的。就其本质来说，可以将图像分为两大类：

一类是模拟图像，包括光学图像、照相图像、电视图像等，例如，在生物医学研究中，人们在显微镜下看到的图像就是一幅光学模拟图像，照片、用线条画的图、绘画也都是模拟图像。模拟图像的处理速度快，但精度和灵活性差，不易查找和判断。

另一类是将连续的模拟图像经过离散化处理后变成计算机能够辨识的点阵图像，称为数字图像。严格的数字图像是一个经过等距离矩形网格采样，对幅度进行等间隔量化的二维函数，因此，数字图像实际上就是被量化的二维采样数组。

本书中涉及到的图像处理都是指数字图像的处理。与模拟图像相比，数字图像具有以下显著优点。

（1）精度高：目前的计算机技术可以将一幅模拟图像数字化为任意的二维数组，即数字图像可以由无限个像素组成，每个像素的亮度可以量化为 12 位（即 4096 个灰度级），这样的精度使得数字图像与彩色照片的效果相差无几。

（2）处理方便：由于数字图像本质上是一组数据，所以可以用计算机对它进行任意方式的修改，例如放大、缩小、改变颜色、复制和删除某一部分等。

（3）重复性好：模拟图像（例如照片）即便是使用非常好的底片和相纸，也会随着时间的流逝而退色、发黄，而数字图像可以长期保存。

1.3.1　图像的数字化

一般的图像都是模拟图像，即图像上的信息是连续变化的模拟量。如一幅黑白灰度照片上的物体是通过照片上各点的光的强度不同而体现的，而照片上的光强是一个连续变化的量，也就是说，在一定的范围内，光强的任何值都可能出现。对于这种模拟图像只能采用模拟处理方式进行处理，例如按光学原理用透镜将照片放大。计算机不能接受和处理模拟信号，只有将图像在空间和灰度上都离散化为数字信号后，或者说将模拟图像变换为数字图像方能接受。为此，常将计算机图像处理称作为数字图像处理。

图像的数字化过程通过采样和量化两步完成。空间坐标的离散化叫做空间采样，灰度的离散化叫做灰度的量化。采样是将时间和空间上连续的图像转换成离散的采样点（即像素）集的过程。事实上，采样就是要决定用多少个点来描述一张图像，采样的结果就是通常所说的图像分辨率。采样可以分为均匀采样和非均匀采样，比较常用的是均匀采样，但是很多情况下可以根据图像特性利用自适应的采样过程来改进图像的视觉效果。例如，在图像灰度过渡区比较尖锐的情况下可以采用较密的采样，而较平滑的区域就可以使用稀疏的采样。采样时要注意采样间隔的选取，采样间隔越小、图像越精细，图像的点越多、其采样数据越大，因而对计算机的负担也就越重。例如，一幅 320×240 的图像，就表示这幅图像是由 76 800 个像素点所组成的。可见，如果想要得到更加清晰的图像效果（也就是使这幅图像具有较高的分辨率），就需要使用更多的点来表示图像，那么相应地就需要更多的存储空间。采样通常使用采样频率来进行标示。采样频率是指一秒钟内采样的次数，它反映了采样点之间的间隔大小。丢失的信息越少，采样频率越高，采出的样本就越细腻、逼真，图像的质量越高，但要求的存储量也越大。

将像素点上的灰度值离散为整数，称之为量化，量化的结果是图像容纳的所有颜色数据。量化决定使用多大范围的数值来表示图像采样之后的每一个点，这个数值范围确定了图像能使用的颜色总数。一般来说，像素的最大灰度级数 G 都取为 2 的整次幂。例如，以 4bit 存储一个点，就表示图像只能有 16 种颜色。数值范围越大，表示图像可以拥有更多的颜色，自然就

可以产生更为细致的图像效果，但是相应地也必须占用更多的存储空间。量化又分为均匀量化和非均匀量化。均匀量化是指简单地在灰度范围内等间隔量化；非均匀量化是对像素出现频度少的部分采用大的间隔，而频度大的部分采用小间隔。通常所说的量化等级，是指每幅图像样本量化后一共可取多少个离散的数值或用多少个二进制数的位来表示，它反映了量化的质量，若每个样本用 8 位（通道数）二进制数表示，则有 2^8（即 256）个量级；若采用 16 位（通道数）二进制数表示，则有 2^{16}（即 65 536）个量级；若采用 24 位（通道数）二进制数表示，则有 2^{24}（即 16 677 万）个量级，同样，量级越大，图像质量就越高，存储空间要求就越大。计算机图像数字化的质量采用 3 个主要参数来进行衡量：采样频率、图像样本量化等级及通道数。

1.3.2 图像的存储

由于计算机的工作速度、存储空间是相对有限的，各种参数都不能无限提高。那么数字化后的图像数据是如何存储的呢?计算机一般采用两种存储方式：一种是位映射（Bitmap），即位图存储模式；另一种是向量处理（Vector），也称矢量存储模式。

位映射是将图像的每一个像素点转换为一个数据，并存放在以字节为单位的矩阵中。当图像是单色时，一个字节可存放 8 个像素点的图像数据；16 色图像每两个像素点用一个字节存储；256 色图像每一个像素点用一个字节存储，这样就能够精确地描述各种不同颜色模式的图像画面。所以此种存储模式较适合于内容复杂的图像和真实的照片，例如，用数码相机和扫描仪获取的图像一般都存储为位图。位图图像的缺点在于随着分辨率以及颜色数的提高，位图图像所占用的磁盘空间会急剧增大，同时在放大图像的过程中，图像也会变得模糊而失真。

向量处理存储图像内容的轮廓部分，而不存储图像数据的每一点。例如，对于一个圆形图案，只要存储圆心的坐标位置和半径长度，以及圆形边线和内部的颜色即可。该存储方式的缺点是经常耗费大量的时间做一些复杂的分析演算工作，但图像的缩放不会影响显示精度，即图像不会失真，而且图像的存储空间较位图方式要少得多。所以，向量处理比较适合存储各种图表和工程设计图。

总体来看，位图是记录每一个像素的颜色值，再把这些像素点组合成一幅图像，而矢量图是保存图形对象的位置和曲线、颜色的算法。位图占用的存储空间较矢量图要大得多，而矢量图的显示速度较位图慢。

1.3.3 数字图像的表示

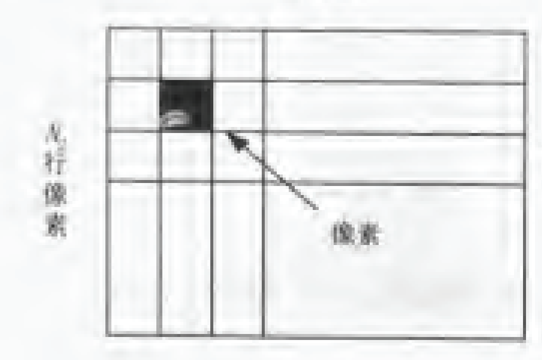
采样量化后的数字图像就是一个灰度值的二维数组。该数组若用 $f(x,y)$ 来表示时，其含义是位于坐标 (x,y) 处的像素，其灰度值是 $f(x,y)$。根据灰度层次及光谱轴与时间轴上组合方式的不同，数字图像分类如表 1-1 所示。表中虽然不同类别的图像，图像的内容和视觉效果都不同，但都可以在计算机内用二维数组的集合表示。因此，研究数字图像的处理，最基本的就是研究一个二维数组的处理。

表 1-1　　数字图像的类别

类　别	表示形式	说　明
二值图像	$f(x, y)=0$ 或 1	文字图像，形态图等，阈值分割后得到
灰度图像	$0\leq f(x, y)\leq 2^n-1$	黑白照片，一般 $n=8$
彩色图像	$\{f_i(x,y)\}$　$i=R,G,B$	RGB 彩色空间的图像
运动图像（时间序列图像）	$\{f_i(x,y)\}$　$i=R,G,B$，　$t=t1,t2,\ldots\ldots,m$	运动视频图像，运动跟踪

一幅 $N_1\times N_2$ 个像素的数字图像，其像素灰度值可用 N_1 列 N_2 行的矩阵 F 来表示，如图 1-2 所示。这样，对数字图像的各种处理就可以变成对矩阵 F 的各种运算。





$$F(x,y)=\begin{bmatrix} f(1,1) & f(1,2) & \cdots & f(1,N_1) \\ f(2,1) & f(2,2) & \cdots & f(2,N_1) \\ \vdots & & & \vdots \\ f(N_2,1) & f(N_2,2) & \cdots & f(N_2,N_1) \end{bmatrix}$$

图 1-2　数字图像的表示

1.4　数字图像处理的基本运算

1.4.1　数字图像处理的基本过程

数字图像信息可看成是一个二维数组 $F(i,j)$，对它处理的基本过程如同电视光栅扫描过程，按照由左到右，由上到下的顺序进行，并在扫描过程中逐点对各像素进行处理。这样的扫描过程称为顺向扫描。与此相对应的，由下到上、由右到左的逆向扫描，也是一种常见的处理过程。这种如同光栅扫描的过程仅仅是图像处理中最基本的处理过程，其他如取某一像素点代表所在区域像素点的稀释处理等，在后续的实际应用中介绍。

1.4.2 基本运算形式

在用计算机对图像进行处理的过程中，有几种基本的但又非常有用的运算形式，它们是后续章节图像处理与识别的基础。下面作简单介绍。

（1）灰度直方图

在数字图像处理中，一种最简单且最有用的工具是灰度直方图。它概括了一幅图像的灰度级内容。任何一幅图像的直方图都包括了可观的信息，某些类型的图像还可由其直方图完全描述。

灰度直方图是灰度级的函数，描述的是图像中具有该灰度级的像素个数；其横坐标是灰度级，纵坐标是该灰度出现的像素个数，如图 1-3 所示。



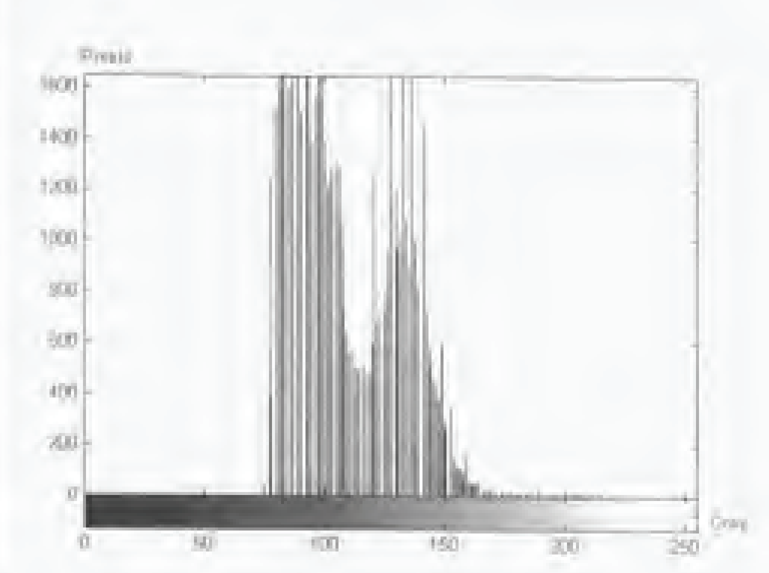


图 1-3　一幅图像及其灰度直方图

通过除以图像的面积来规一化灰度直方图可得到图像的概率密度函数（PDF），对面积的函数进行规一化处理可得到图像的累积分布函数（CDF）。这在图像的统计处理时非常有用。借助于灰度直方图可以方便地进行图像均衡处理、阈值分割等操作。

（2）点运算

在图像处理过程中，点运算是一种简单但很重要的技术，它能改变图像像素点的灰度范围，当显示一幅图像时，点运算的作用尤其明显。

对于一幅输入图像，经过点运算将产生一幅输出图像，后者的每个像素点的灰度值仅由相应输入像素点的值决定，如图 1-4 所示。因此，点运算不可能改变图像内的空间关系。

点运算以预定的方式改变一幅图像的灰度直方图。除灰度级的改变是根据某种特定的灰度变换函数进行之外，点运算可以看作是“从像素到像素”的复制操作。像素之间不发生关系，各像素的处理独立进行。若输入图像为 $A(x,y)$，输出图像为 $B(x,y)$，则点运算可以表示为：

$$B(x,y)=f[A(x,y)] \tag{1-1}$$

点运算可完全由灰度变换函数$f(D)$确定，它描述了输入灰度级和输出灰度级之间的映射关系。

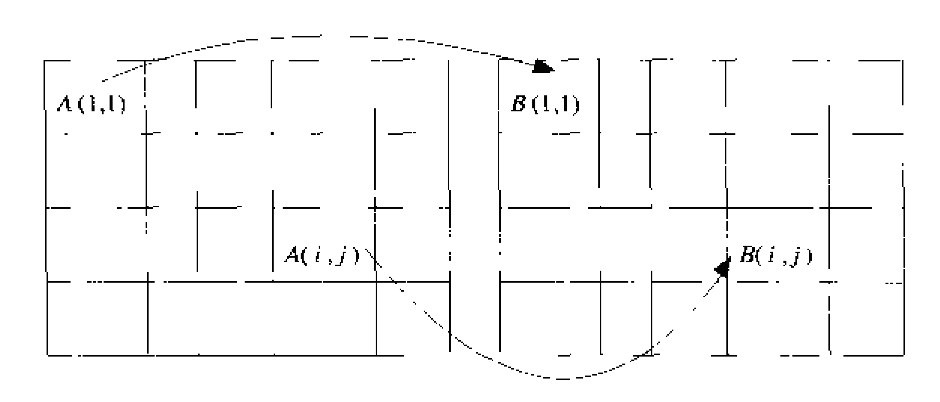


图1-4 点运算

典型的点运算为光度学标定，通过对图像传感器的非线性特性作出补偿来反映某些物理特性，如光照强度等；对比度增强，调整图像的亮度、对比度，以便观察，以及显示标定，突出用户感兴趣的特征等。

（3）代数运算

代数运算是指对两幅图像进行点对点的加、减、乘、除计算得到的输出图像的运算。图像处理的4种代数运算的数学表达式如下：

$$C(x,y)=A(x,y)+B(x,y) \tag{1-2}$$

$$C(x,y)=A(x,y)-B(x,y) \tag{1-3}$$

$$C(x,y)=A(x,y)\times B(x,y) \tag{1-4}$$

$$C(x,y)=A(x,y)\div B(x,y) \tag{1-5}$$

其中$A(x,y)$和$B(x,y)$为输入图像，而$C(x,y)$为输出图像。还可以通过适当的组合，形成涉及几幅图像的复合代数运算。

图像的代数运算在图像处理中有着广泛的应用。例如图像相加可以对同一场景的多幅图像求平均值，有效地抑制随机噪声的影响，还可以将一幅图像的内容叠加到另一幅图像上去，以达到增强效果的作用。图像相减运算可以对某一场景中的序列图像进行运动检测和控制。两幅图像进行乘法运算可以实现掩模操作，即屏蔽掉图像的某些部分。

（4）几何运算

几何运算可以改变图像中各物体之间的空间关系。这种运算可以看成是将物体在图像内移动。该移动过程可以改变图像中的物体像素之间的关系。几何运算可以是不受任何限制的，但是通常都需要做一些限制以保持图像的外观顺序。一个几何运算需要两个独立的算法。首先，需要一个算法来定义空间变换本身，用它描述每个像素如何从其初始位置"移动到终止位置"，即每个像素的"运动"；另外，还需要一个用于灰度级插值的算法，因为在一般情况下，输入图像的位置坐标(x,y)为整数，而输出图像的位置坐标为非整数。

空间变换主要用来保持图像中曲线的连续性和物体的连通性，其数学形式描述可以定义

为：

$$g(x,y)=f(x',y')=f[a(x,y),b(x,y)] \tag{1-6}$$

其中，f 表示输入图像，g 表示输出图像，坐标（x',y'）指的是空间变换后的坐标。$a(x,y)$ 和 $b(x,y)$ 分别是图像的 x 和 y 的空间变换函数。

灰度插值主要是对空间变换后的像素赋予灰度值。图像一般用整数位置处的像素来定义，但在几何变换中，$g(x,y)$ 的灰度值一般由处在非整数坐标上的 $f(x,y)$ 的值来确定，所以必须通过灰度插值实现。

几何运算可以用来消除由于摄像机导致的数字图像的几何畸变，通过对相似图像的配准进行图像的比较等。

1.5 图像处理与识别及图像理解所研究的内容

数字图像处理和识别学科所涉及的知识非常广泛，具体的方法种类繁多，应用也极为普遍，但从学科研究内容上可以分为以下几个方面。

1.5.1 图像处理技术

1. 图像数字化

其目的是将模拟形式的图像通过数字化设备变为数字计算机可用的离散的图像数据。

2. 图像变换

为了达到某种目的（通常是从图像中获取某种重要的信息）而对图像使用一种数学技巧，经过变换后的图像更为方便、容易地处理和操作。

3. 图像增强

图像增强的主要目标是改善图像的质量。采用某种特殊的技术来突出图像中的某些信息，削弱或消除某些无关信息，从而有目的地强调图像的整体或局部特征。常常用来改善人对图像的视觉效果，让观察者能看到更加直接、清晰、适于分析的信息。直方图修正、灰度变换、强化图像轮廓等都是常用的手段。

4. 图像分割

在图像研究和应用中，人们往往仅对图像的某些部分感兴趣。它们一般对应图像中特定的、具有独特性质的区域。图像分割就是把图像分割成各具特性的区域并提取出感兴趣的目标。

5. 图像分析

图像分析也可以称为图像理解，主要研究从图像中提取有用的数据或信息，生成非图像的描述或表示。图像分析的内容分为特征提取、图像分割、符号描述、纹理分析、运动图像分析和图像的检测与配准。

1.5.2　图像识别技术

图像识别是近20年来发展起来的一门新型技术科学，它以研究某些对象或过程（统称图像）的分类与描述为主要内容。

图像识别所研究的领域十分广泛，它可以是医学图像中的癌细胞识别；机械加工中零部件的识别、分类；可以是从遥感图片中辨别农作物、森林、湖泊和军事设施，以及判断农作物的长势，预测收获量等；可以是自导引小车中的路径识别；邮政系统中自动分拣信函；交通管制、识别违章行驶的汽车牌照；银行的支票识别、身份证识别等。上述都是图像识别研究的课题。总体来说所研究的问题，主要是分类问题。

1. 图像识别系统的组成

一个图像识别系统主要包括3部分：①图像信息获取；②信息加工和处理、抽取特征；③判断或分类等，如图1-5所示。

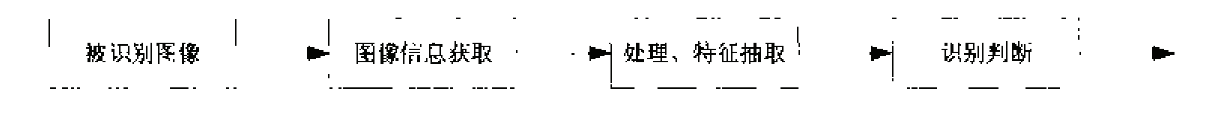


图1-5　图像识别系统框图

2. 图像识别方法

图像识别方法较多，大体上可以归纳为两类方法：统计方法（数学方法）和语言（或结构）学方法，后者亦称句法结构识别方法。

统计方法以数学上决策理论为基础，根据这种理论建立了统计学识别模型。其基本模型是在对研究的图像进行大量统计分析，找出规律性认识，抽出反映图像本质特点的特征进行识别。在这种方法中，大量工作在于如何抽取图像的特征或决定统计参数，即所谓参数法。另外，还有非参数决策法，如近邻法则，它是一种绕过概率的估计而直接进行决策的方法。对于特征抽取，必须把图像的大量原始信息缩减为少数的特征，例如采用方差分布、特征向量法等。对文字、符号等可只抽取几何形状特征，对声波信号可抽取频谱特征。为了抽取特征，有时要对原始图像信息进行各种变换，空间投影，把多维的图像点简化到几个坐标分量上。例如，在高空用多波段遥感仪得到的遥感照片，具有大量的图像数据，为了进行识别，可先将其划分成若干小的集群，将性质相近的数据点划为一个集群，进行聚合分析。如利用梯度法反复迭代计算，可把数据点的距离小于某一数值的点合并在一起。从而大大减少信息量，只需研究这些集群的性质就够了，这就是集群分析。

句法结构识别法立足于分析图像的结构，一幅图像可以模仿语言构造，用一些语句来表达。语句的结构是由词、短语等组成，并按一定的语法表达出来。也就是说，语句由短语组成，而短语由单词组成，其中最基本的元素是单词。那么一些语句又怎样和图像发生联系呢?这可从图像的形成谈起，任何一幅图像，总是由一些点、直线、斜线、弧线及环等组成，剖析图像的这些基本元素，看它们按怎样的规则构成图像，这就是结构分析的课题。这些基本元素就相当于语句中的单词；那些直线、曲线的某种组合可看成短语，它们的全体按怎样的规则构成整个

图像，就相当于语法规则，而对图像识别来说，就相当于检查图像所代表的某一类句型，是否符合事先规定的语法。若语法正确，则认为识别出结果。

由上述可知，这种方法主要是利用了图像结构上的相互关系。这种语言学方法起始于 20 世纪 60 年代后期，发展较晚，在实用中还有一些问题。例如由于图像比语言要复杂得多，语言中的词是一个接一个的一串符号排列，而要让图像的基本元素也排成一串，就不容易了。因为图像的基本元素，其结构关系是上下左右交叉一起的，这就需要合理选择和设计基本元素。

综上所述，两类方法各有优缺点。第一类方法很少利用图像本身的结构关系，而第二类方法则没有考虑图像在环境中所受的噪声干扰，必然使其元素或结构关系带有一定的随机性。因而，把二者结合起来，各取其长，是可取的途径。例如研究具有随机性质的语言学模型就很有必要，而具有学习能力的语言学模型即可认为是其一例。

在图像识别技术中，从识别逻辑的观点来看，亦可分为两个类型：组合式的和顺序式的。前者是把图像的特征全部抽出（或足以判别一个图像的很大一部分特征）之后再进行判断，给出结果。后者则按所抽出特征的次序，每抽出一次特征，都要进行一次判断（不是对整个图像），直到最终给出结果。这类逐步判断的方式，可认为是一种多阶段最优问题。动态规划法是这类顺序识别的实例。

除了这两类识别外，近年来模糊数学和神经网络在图像识别领域中也得到了广泛应用。有很多问题利用模糊数学或神经网络的概念进行识别可以获得快速准确的结果，本书将在“模糊集识别方法”和“神经网络识别方法”中做相关介绍。

3. 文字识别

“文字”是人们相互交流信息的主要工具。从图的角度来看，它是简单的图形；从技术上看，文字图形的识别相对来说比较简单，易于处理和分类。因此，在图像识别中，文字识别的研究比较深入广泛，取得的成果也较大。在国际上，文字识别目前主要指的是光学文字识别（OCR）。一般是识别字母、数字和符号 3 种。文字识别技术广泛应用于阅读邮件地址、对邮件进行自动分拣及识别银行支票、购货账单、贸易表格等。也可用于管理报告、文件检索、语言翻译、资料分析等；在医学诊断过程和数字通信终端中也有其应用等。但其最主要的一个用途还是作为电子计算机的输入设备，用以代替人或键盘的工作，自动地把文字和其他信息送往计算机。

文字识别装置的结构框图如图 1-6 所示。图中光电变换检测部分的主要功能是对纸面上的文字进行光电转换，然后将信号送往其后的各部分进行处理和识别。常用的检测设备有飞点扫描器、光敏器件矩阵、激光扫描器等，预处理部分的功能是将已变成电信号的信息加以处理，去除信号中的污点、空白等噪声，并根据一定准则除掉一些非本质信号，对文字的大小、位置和笔划粗细等进行规范化，以便简化判断部分的复杂性。特征抽取部分是从整形和规范化的信号中抽取反映字符本质的有用信息，供识别部分进行识别。作为特征被抽取的内容是较多的，可以是几何特征，如文字线条的端点、折点和交点等。识别判断部分则是根据抽取的特征，运用一定的识别原理，对文字进行分类，确定其属性，达到识别的目的，实际上判断部分就是一个分类器。

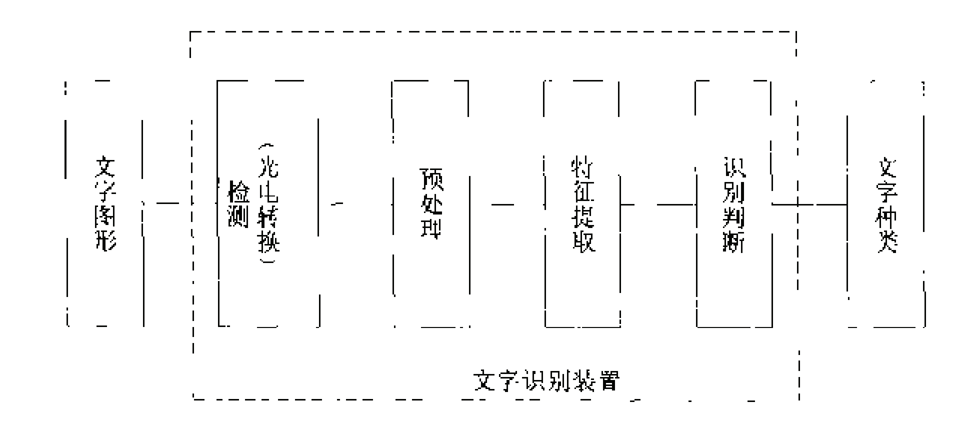


图1-6 文字识别装置框图

文字识别的识别原理可分为相关匹配识别原理、概率判定准则及句法模式识别3大类。实际应用时，常常将几种结合使用以取得较高的准确性。

1.5.3 图像理解

上述图像处理与图像识别的最终目的，就在于对图像的描述和解释，以便最终理解它是什么图像。所以它是在图像处理及图像识别的基础上，再根据分类作结构分析，去描述图像和解释图像。因而图像理解包括图像处理、图像识别和结构分析。它是结合了人工智能与专家系统而发展起来的一门新兴学科。

图像理解研究用计算机系统解释图像，实现类似人类视觉系统理解外部世界的一门科学。所讨论的问题是，为了完成某一任务，需要从图像中获取哪些信息，以及如何利用这些信息获得必要的解释。图像理解的研究显然要涉及或包含研究获取图像的方法、装置和具体应用的实现，这就形成了所谓计算机视觉。

图像理解系统有几个特点：

（1）分阶段的信息处理带来了信息的多层表示；

（2）对图像的解释，是以某种形式的描述实现的；

（3）图像的正确解释离不开知识的导引，这一特点对人的视觉理解也适合，即经验对视觉理解的重要性。

图像理解的目的是解释一幅图像，这必然要涉及到识别图像中的目标，说明图像中各目标之间的关系，测量出各目标的相对尺寸和离观察点（如相机）的距离等，其中目标的描述是一项主要内容。若仅仅考虑利用景物的二维投影时，利用区域（或边界）及其属性就可以描述目标。而若考虑三维情况，就要涉及三维目标的几何表示问题。

图像理解系统在设计上应该考虑的有以下几个重要问题：

（1）从图像中提取哪些信息；

（2）这些信息在图像中怎样表示；

（3）在理解过程中计算机必须具有哪些知识；

（4）最合适的图像处理程序是什么；

（5）知识以及各个阶段图像处理的结果如何表示。

为了简单说明，我们可以用图1-7来表示一个图像理解系统。

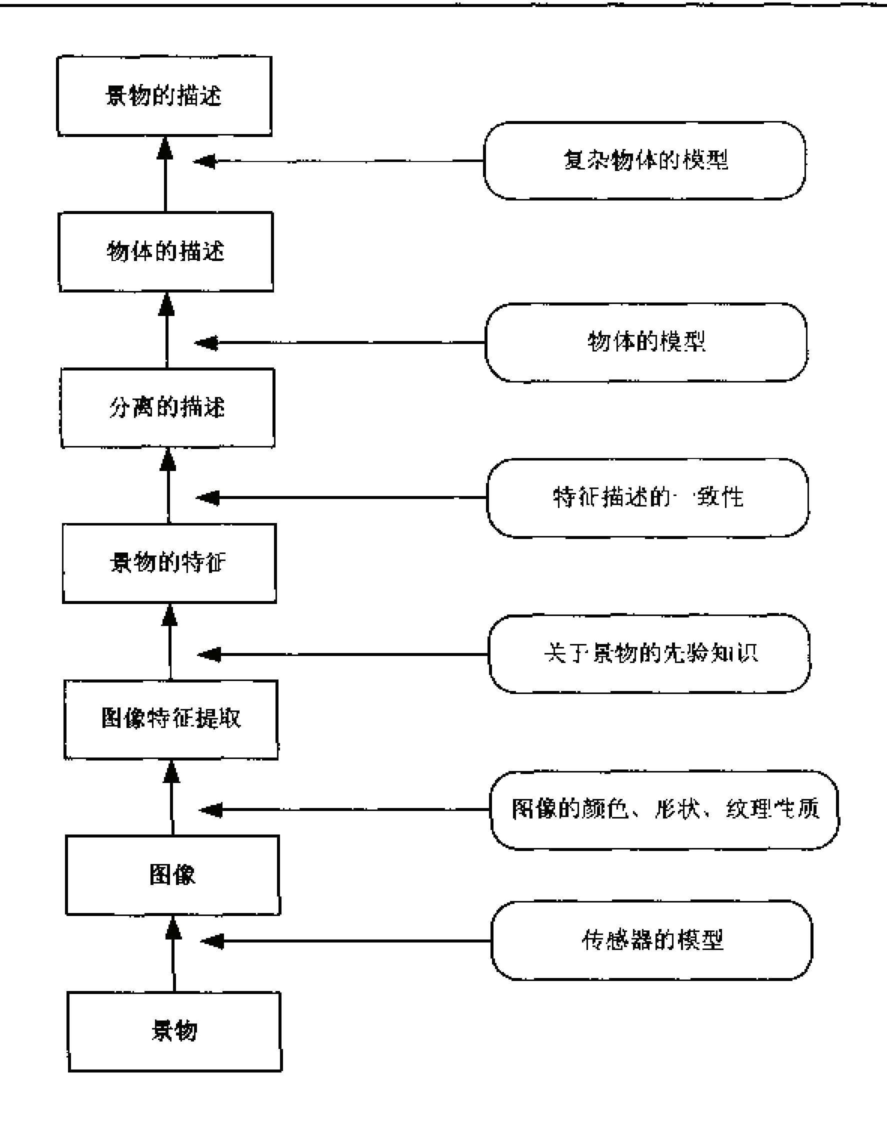


图 1-7 典型的图像理解系统

1. 传感器引起失真的补偿

人的图像是用灰度的二维阵列表示的，通常由于传感器的特性会引起几何学上或者光学上的失真，因此需进行补偿。

2. 图像特征的提取

图像凡是亮度发生急剧变化的地方，都是对应于不同物体面与面之间的边缘，这是一个重要特征，所以把图像中亮度急剧变化的点提取出来并对其性质进行描述（如边缘的方向、幅度、直方图等），对这些特征点的统计分布状况进行研究，可认识到物体的构造。如果把相邻的特征点连接起来则可以构成输入图像的曲线。

3. 景物特征的提取

在观看物体时看到的不仅是亮度急剧变化的点，同时还有各个面的方向、距离、颜色反射率，这些特征与照明的情况和视点的位置无关，是物体所固有的特征。为了与图像上的特征相区别，一般称为景物特征。

4. 景物的分割与物体的发现

利用景物的特征，把其性质大致相同的领域分割开来，这与图像处理中把亮度相同的领域分割出来的领域法相同。

5. 物体的识别

从景物所发现的各物体再参照物体的模型来识别物体，与此同时确定各物体的位置和方向。

6. 景物的识别

研究各物体之间的连接关系，由这个连接关系和模型来认识更复杂的物体，从而可以对景物的全体进行描述。

1.6　图像处理与识别及图像理解的关系

在研究图像理解时，往往要进行图像的处理、识别及图像理解的过程，三者关系非常密切，互相交错。下面就图像处理、识别及图像理解的过程、作用及相互关系进行讲述。

1.6.1　图像处理

在研究图像时，首先要对获得的图像信息进行预处理（前处理）以滤去干扰、噪声，作几何、彩色校正等。这样可提高信噪比；有时由于信息微弱，无法辨识，还得进行增强处理。增强的作用，在于提供一个满足一定要求的图像，或对图像进行变换，以便人或计算机分析。并且为了从图像中找到需要识别的东西，还得对图像进行分割，也就是进行定位和分离，以分出不同的物体。为了给观察者以清晰的图像，还要对图像进行改善，即进行复原处理，它是把已经退化了图像加以重建或恢复的过程，以便改进图像的保真度。在实际处理中，由于图像信息量非常大，在存储及传送时，还要对图像信息进行压缩。

上述工作必须用计算机进行，因而要进行编码等工作。编码的作用，是用最少数量的编码位（亦称比特），表示单色和彩色图像，以便更有效地传输和存储。

以上所述都属图像处理的范畴。因此，图像处理包括图像编码、图像增强、图像压缩、图像复原、图像分割等。对图像处理环节来说，输入是图像，输出也是图像，也就是处理后的图像，如图 1-8 所示。由图像处理的内容可见，图像处理的目的主要在于解决两个问题：一是判断图像中有无需要的信息；另一是确定这些信息是什么。例如，机械零部件识别的数据信息有：

（1）零部件的亮度和色度信息；

（2）零部件的形状信息；

（3）纹理信息；

（4）尺寸信息；

（5）几何精度信息等。

抽取这些有用信息的主要目的在于改善图像质量和进行图像识别。

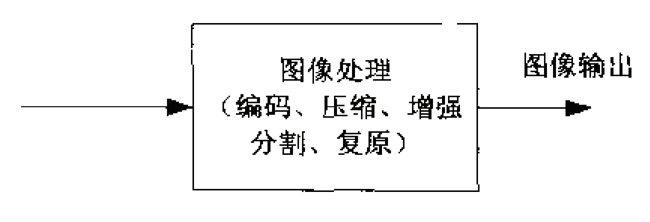


图 1-8　图像处理示意图

图像识别对上述处理后的图像进行分类，确定类别名称，它可在分割的基础上选择需要提取的特征，并对某些参数进行测量，再提取这些特征；最后根据测量结果作分类。为了更好地识别图像，还要对整个图像作结构上的分析，对图像进行描述，以便对图像的主要信息得到一个解释和理解，并通过许多对象相互间的结构关系对图像加深理解，更好地帮助识别。所以图像识别是在上述分割后的每个部分中，找出它的形状及纹理等特征，即特征抽取（有时也包括图像分割），以便对图像进行分类，并对整个图像作结构上的分析。因而对图像识别环节来说，输入是图像（一般是经过上述处理过的图像），输出是类别和图像的结构分析，如图 1-9 所示。而结构分析的结果则是对图像作描述，从而对图像的重要信息得到一种理解和解释。

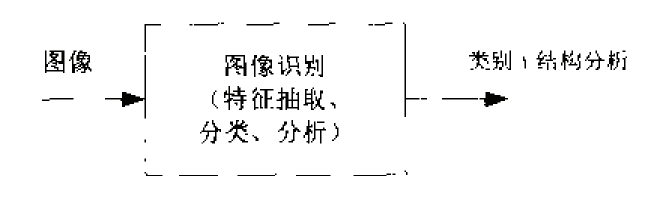


图 1-9　图像识别示意图

这里要注意的是，图像分割不一定完全在图像处理时进行，对有些问题，一面进行分割，一面进行识别，如机械零件的分拣、分档就是如此。所以说，图像处理和图像识别是相互交叉的。

1.6.2　图像理解

所谓图像理解是一个总称。上述图像处理及图像识别的最终目的，就在于对图像作描述和解释，以便最终理解它是什么图像。所以它是在图像处理及图像识别的基础上，再根据分类作结构句法分析，去描述图像和解释图像。因而图像理解包括图像处理、图像识别和结构分析。对理解部分来说，输入是图像，输出则是图像的描述与解释，如图 1-10 所示。图像处理、识别及理解之间的关系见表 1-2。

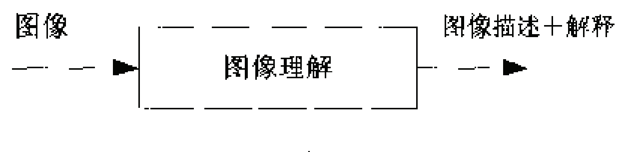


图 1-10　图像理解示意图

表 1-2　　图像处理、识别及理解的关系

输入 / 输出	图　像	描　述
图像	图像处理	计算机“图示学”（“作图”）
类别及结构分析	图像识别	其他
描述及解释	图像理解	其他

实质上，图像理解属于人工智能的范畴。图像理解也要作图像处理、识别及结构分析。例

如计算机下棋。首先要把人的“智慧”存储在计算机中，计算机在接受了一部分“智慧”后，便能根据逻辑推理进行分析、推断等工作。

1.6.3　图像识别与图像处理及图像理解的关系

如上所述，图像理解是一个总称，它是一个系统，其一般概念可用图1-11说明。其中每一部分和其前面的一部分都有一定的关系，也可以说有一种反馈作用，例如分割可以在预处理中进行。并且，该系统不是孤立的，为了发挥其功能，它时时刻刻需要来自外界的必要信息，以便使每个部分能有效地工作。这些外界信息是指处理问题及解决问题的看法、设想、方法等，例如，根据实际图像，在处理部分需要采用什么样的预处理，在识别部分需要怎样分割，抽取什么样的特征及怎样抽取特征，怎样进行分类，要分多少类，以及最后提供结构分析所需的结构信息等。

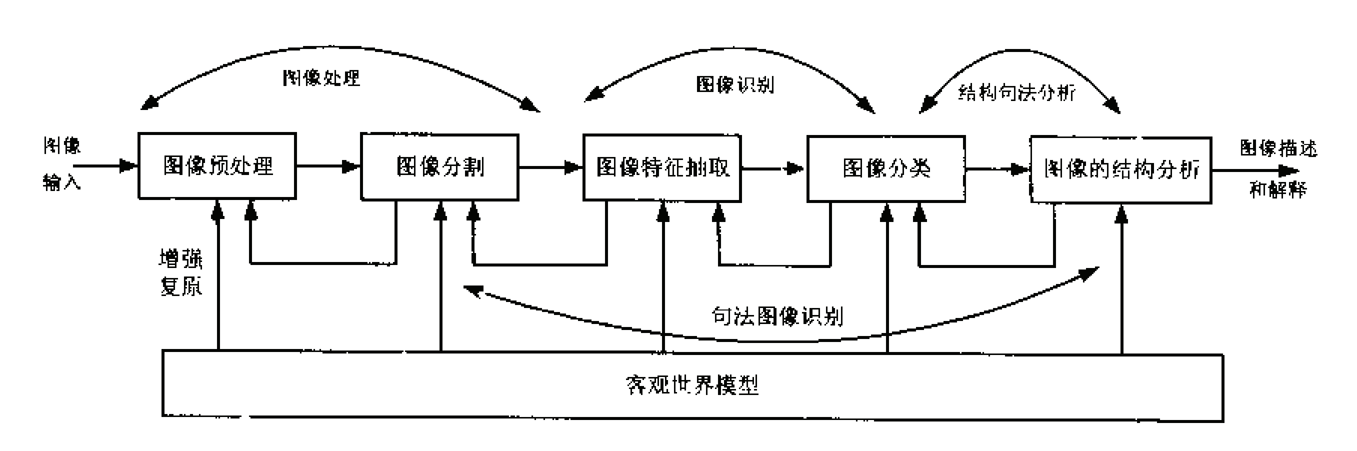


图1-11　图像理解系统关系图

在该系统中，图像预处理、图像分割为图像处理；图像特征提取和图像分类属图像识别；而结构句法分析涉及的内容则是从图像分割到图像结构分析这一过程；整个系统所得到的结果是图像的描述及解释。当某个新的对象（图像）送进系统时，就可以进行解释，说明它是什么。

1.7　计算机视觉

人类在推动社会进步的过程中，面临着自身能力、能量的局限性，因而发明和创造了许多机器来辅助或代替人类完成任务。智能机器，包括智能机器人，是这种机器最理想的形式，也是人类科学研究中所面临的最大挑战之一。智能机器是指这样一种系统，它能模拟人类的功能，能感知外部世界并有效地解决人所能解决的问题。因此，对于智能机器来说，赋予机器以人类视觉功能对发展智能机器是极其重要的，也由此形成了一门新的学科——计算机视觉（也称机器视觉或图像分析与理解等），它是人工智能领域最热门的研究课题之一。

计算机视觉是研究用计算机来模拟生物外显或宏观视觉功能的科学和技术。计算机视觉系统的首要目标是用图像创建或恢复现实世界模型，然后认知现实世界。计算机视觉系统获取的场景图像一般是灰度图像，即三维场景在二维平面上的投影。此时，场景三维信息只能通过灰度图像或灰度图像序列来恢复处理，这种恢复需要进行多点对一点的映射逆变换。在信息恢复

过程中，还需要有关的场景知识和投影几何知识。

计算机视觉是一个相当新且发展十分迅速的研究领域，并成为计算机科学的重要研究领域之一。它和专家系统、自然语言理解已成为人工智能最活跃的三大领域。尽管它还是一门“年轻”的学科，还没有形成完整的理论体系，在很多方面它解决问题的方法还是一种技巧，但它是实现工业生产高度自动化、机器人智能化、自主车导航、目标跟踪，以及各种工业检测、医疗和军事应用的核心内容之一，也是实现智能机器人的关键因素之一。对于什么是计算机视觉，计算机视觉研究什么，与哪些学科相关，支撑的硬件是什么及发展的现状如何等一系列问题应有明确的认识，本节从这些方面作一些介绍。

1.7.1 计算机视觉研究的内容

计算机视觉研究可以分为如下 5 大研究内容。

1. 输入设备

输入设备（Input Device）包括成像设备和数字化设备。成像设备是指通过光学摄像机或红外、激光、超声、X 射线对周围场景或物体进行探测成像，得到关于场景或物体的二维或三维数字化图像。获取数字化图像是计算机视觉系统的最基本的功能。目前用于视觉研究的大多数输入设备是商品化的产品，如 CCD 黑白或彩色摄像机、数字扫描仪、超声成像探测仪、CT 成像设备等。但这些商品化的输入设备远远不能满足实际的需要，因此，仍有许多研究人员在研究各种性能先进的成像系统，如红外成像系统、激光成像系统，还有所谓的计算成像系统（Computational Imaging），即每一个像素元（或若干像素元）对应一个简单的处理器，这样可以适应复杂变化的动态场景。

2. 低层视觉

低层视觉（Low Level）主要是对输入的原始图像进行处理。这一过程借用了大量的图像处理技术和算法，如图像滤波、图像增强、边缘检测等，以便从图像中抽取诸如角点、边缘、线条、边界以及色彩等关于场景的基本特征。这一过程还包含了各种图像变换（如校正）、图像纹理检测、图像运动检测等。

3. 中层视觉

中层视觉（Middle Level）的主要任务是恢复场景的深度、表面法线方向、轮廓等有关场景的 2.5 维信息，实现的途径有立体视觉（Stereo Vision）、测距成像（Range Finder）、运动估计（Motion Estimation）、明暗特征、纹理特征等所谓的从 X 恢复形状（Shape from X）的估计方法。系统标定、系统成像模型等研究内容一般也是在这个层次上进行的。

4. 高层视觉

高层视觉（High Level）的任务是在以物体为中心的坐标系中，在原始输入图像、图像基本特征、2.5 维图的基础上，恢复物体的完整三维图，建立物体三维描述，识别三维物体并确定物体的位置和方向。

值得指出，低层、中层和高层视觉基本上与 Marr 视觉的 3 个阶段相对应。另外，主动视觉（Active Vision）涵盖了上述各个层次的研究内容。

5. 体系结构

体系结构（System Architecture）这一术语通常的含义指在高度抽象的层次上，根据系统

模型而不是根据实际设计的具体例子来研究系统的结构。为了说明这一点，可以考虑建筑设计中某一时期的建筑风格和根据这一风格设计出来的具体建筑之间的区别。体系结构研究涉及一系列相关的课题：并行结构、分层结构、信息流结构、拓扑结构以及从设计到实现的途径等。

1.7.2 计算机视觉发展的现状

自从20世纪70年代末，马尔提出他的视觉计算理论以来，其观点逐步为大多数计算机视觉研究者所接受，并成为这一领域的主导思想。30 年来，在马尔的理论框架下，计算机视觉取得了一大批成果，诸如立体视觉中外极线的约束，运动分析中光流的基本方法及其后的改进等。另外，也包括数据结构和算法层次上的各种算法，还包括在硬件实施方面一些试验系统，特别是在早期视觉方面。

然而，20世纪80年代曾给人们以很大希望的马尔视觉计算理论在实际应用时遇到了困难。主要困难在于三维景物分析方面，即从景物图像或系列图像精确求出景物的三维几何描述，并定量地确定景物中物体的性质。所以最近一段时间，人们对马尔这一理论展开了不同意见的讨论。马尔视觉计算理论具有以下特点：比较系统地和普遍性地揭示了用二维图像恢复三维物体形态的可能性和基本方法；突出了通用性，特别是早、中期视觉；突出了定量地恢复物体的形状和位置，使相应的视觉问题变得非常复杂和困难；在信息传输路线上是单向的，没有反馈回路。在生物视觉系统中，发现了有许多从高层次传送信息的神经纤维，甚至视网膜上也有许多来自中枢的神经，给予支配信息。虽然上述反馈神经的确切作用尚不明确，但是可以证明，视觉系统应该有反馈存在；计算理论框架固定，与识别功能是从系统进化和个体学习而得到的情况不符合。

面对马尔视觉计算理论的困惑，人们对计算机视觉曾产生过分的悲观和失望，如1986年法国召开的第8届国际模式识别会议上，Parlolis 认为，图像处理领域最近15年进展不大。他提出了不同于马尔的视觉计算模型，诸如，有目的的视觉、面向任务的视觉、主动视觉、定时视觉等。1990年在美国召开的第10届模式识别会议上，J.Atoimonos（英国马里兰大学）提出了有目的的、定性的主动视觉。他说，计算机视觉比人的视觉能力相差甚远，主要原因是：抽取有用的视觉信息涉及太多的计算；计算的稳定性、可靠性太差；视觉系统在恢复客观世界时包含了许多不必要的东西。

定性主动视觉回答的都是一些主要问题，有目的，且稳健。Jain 和 Binford 认为，现在计算机视觉所应用的研究方法不对，知识利用不够，没有充分考虑时空约束和缺少实验分析和验证。Atoimonos 和 Aesenfeld 则认为，以马尔视觉计算理论为代表的理论将视觉规定为由场景的图像精确地获得三维几何计算机视觉来说太高了，是不必要的，计算也太复杂，求解太困难，而这些定量计算没有必要，可用定性视觉来代替定量视觉。

所有这些，最主要的是强调结合具体对象引入知识的重要性，他们认为：任何一个视觉感知系统，都依赖于各种形式的知识，无论是生物还是机器感知系统，其目的都是为了构造现实世界的模型，并用这个模型与物理世界发生相互作用。在构造一个个模型的过程中，系统要用到物理知识、传感知识和该系统应用领域中的一般知识等。

1.7.3 计算机视觉面临的困难

人们对上述几个研究内容进行了卓有成效的研究，研究出大量的技术和算法，并且在各个

领域中得到广泛的应用。不过，机器视觉技术仍处于十分不成熟的阶段。

对于人类视觉来说，识别和理解周围场景是一件非常容易的事，但对于机器来说，却是一件很困难的事，主要困难体现在如下几方面。

1. 图像多义性

三维场景被投影为二维图像，深度和不可见部分的信息被丢失，因而会出现不同形状的三维物体投影在图像平面上产生相同图像的问题。另外，在不同视角获取同一物体的图像也会有很大的差异。

2. 环境因素影响

场景中的诸多因素，包括照明、物体形状、表面颜色、摄像机以及空间关系变化都会对生成的图像有影响，因此，当任何一个因素发生变化时，都会对图像产生影响。

3. 知识导引

同样的图像在不同的知识导引下，将会产生不同的识别结果。同一物体从不同的视角拍摄图像时所反映的图像内容差别很大，因此不同的知识导引可能产生不同的空间关系，图像所表达的也就具有的意义。

4. 大量数据

灰度图像、彩色图像和深度图像的信息量都十分巨大，比如分辨率为 512×512 的灰度图像的数据量为 256kB，同样分辨率的彩色图像的数据量是 768kB。如果处理的是图像列，则数据量更大。巨大的数据量需要巨大的存贮空间，同时不易实现快速处理。

为了解决视觉所面临的问题，研究人员不断寻求新的途径和手段，比如，主动视觉（Active Vision），面向任务的视觉（Task-Oriented Vision），基于知识、基于模型的视觉，以及多传感器信息融合和集成视觉等方法。

第 2 章　MATLAB 语言图像编程

MATLAB 是 Matrix Laboratory 的缩写，它是一种直译式的语言，与其他一些程序设计语言相比，使用上容易多了。它的主要功能是做矩阵的数值运算。它的数值分析、模拟与运算功能也非常强大，而且程序结构完整，又具有很强的平行移植性。MATLAB 目前已广泛流行于图像处理、自动控制、语音处理、生物医学工程、信号分析等各个领域。

本章主要简单介绍 MATLAB 语言的基本操作，编程基础以及如何使用 MATLAB 语言简单处理图像、图像类型转换以及显示图像。

2.1　MATLAB 基本操作

2.1.1　操作界面的默认外形

MATLAB 安装完成后，运行该程序，可以得到如图 2-1 所示的操作界面的默认外形。该界面的上层铺放着 3 个最常见的界面：指令窗、交互界面分类目录和历史指令窗。在默认情况下，还有两个只能看到“窗名”的常用交互界面：工作空间浏览器和当前目录窗。

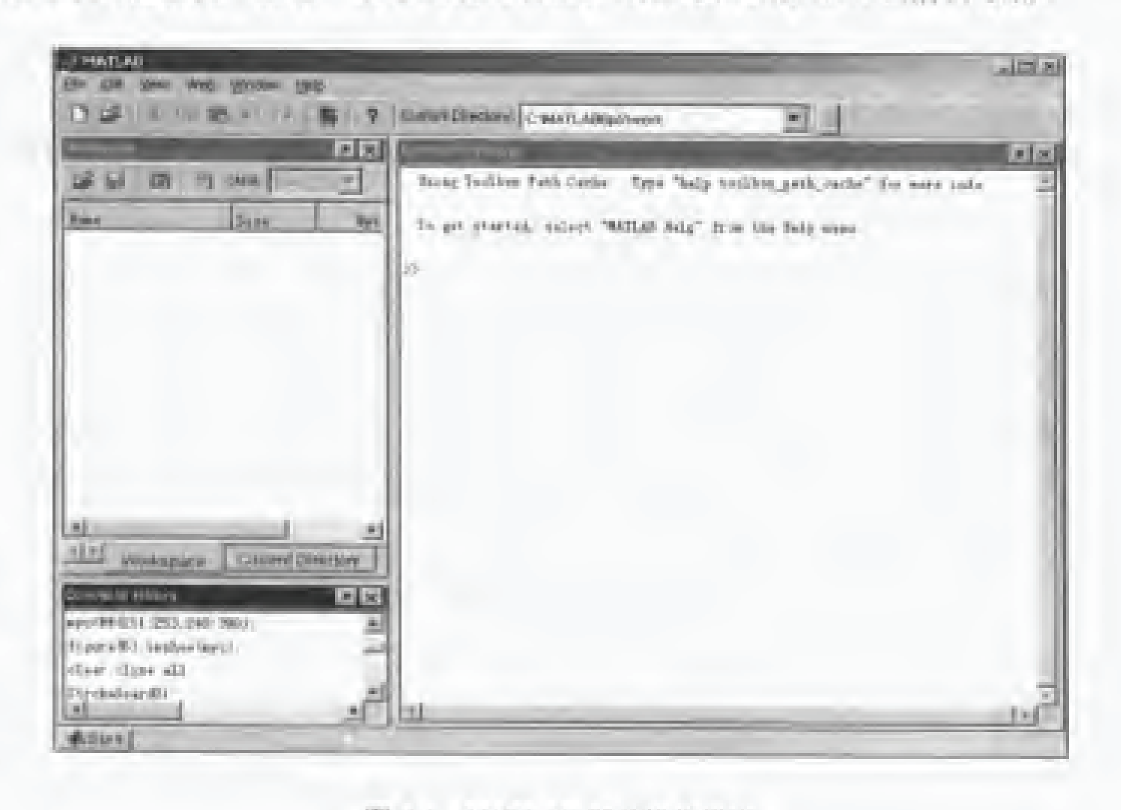


图 2-1　MATLAB 默认操作界面

2.1.2 通用操作界面

下面所列的 8 个交互界面最常用。

（1）指令窗（Command Window）

该窗缺省地处在 MATLAB 界面的右侧。该窗是进行各种 MATLAB 操作的最主要窗口。在该窗内，可键入各种送给 MATLAB 运作的指令、函数、表达式，并显示除图形外的所有运算结果。

注意：输入的指令必须是在英文状态的字母，否则 MATLAB 不接受指令。

（2）历史指令窗（Command History）

该窗缺省地处于 MATLAB 界面的左下侧前台。该窗记录已经运作过的指令、函数、表达式；允许用户对它们进行选择复制、重复运行，以及产生 M 文件。

（3）当前目录浏览器（Current Directory Browser）

该浏览器缺省地位于 MATLAB 界面左下侧的后台。在该交互界面中，可以进行当前目录的设置；展示相应目录上的 M、MDL 等文件；复制、编辑和运行 M 文件；装载 MAT 数据文件。

（4）工作空间浏览器（Workspace Browser）

该交互界面缺省地位于 MATLAB 界面的左上侧后台。该窗口罗列出 MATLAB 工作空间中所有的变量名、大小、字节数。在该窗中，可对变量进行观察、编辑、提取和保存。

（5）内存数组编辑器（Array Editor）

在缺省情况下，该编写器不随操作界面的出现而启动。只有当在工作空间浏览器中对变量进行操作时才启动。

（6）交互界面分类目录窗（Lauch Pad）

该窗口缺省情况下，一般不打开。在 View 下选中 Laud Pad 后，它处在 MATLAB 界面的左上侧后台。该窗以可展开的树状结构罗列着 MATLAB 提供的所有交互界面，包括帮助界面、演示界面、各种应用交互界面。用户若“双击”树状结构上的分类图标，就可展现出相应的交互界面。

（7）M 文件编辑/调试器（Editor/Debugger）

在缺省情况下，该编辑/调试器不随操作界面的出现而启动。只有当进行“打开文件”等操作时，该编辑/调试器才启动。

（8）帮助导航/浏览器（Help Navigator/Browser）

该浏览器缺省情况下，并不随操作界面的出现而启动。只有当作了一定的选择或设置的情况下，才以独立交互界面的形式出现。该浏览器展示由超文本写成的详尽在线帮助。

2.2 MATLAB 编程基础

2.2.1 变量

MATLAB 对变量名称有以下 3 个规定：

（1）变量名称开头必须是英文字母，后面可以接英文字母、下划线、数字；

（2）区分大小写；

（3）变量名称长度应不超过 31 个字符。

MATLAB 中的变量可以声明为局部变量或全局变量，在一个函数内部的变量如无特殊声明，那么这个变量为局部变量，即只能在该函数内部使用；如果希望两个或以上的函数共用某变量，那么应使用 Global，将其定义为全局变量。在程序中使用全局变量可以减少参数的传递，如果能够合理地使用全局变量，将会提高程序的执行效率。

在 MATLAB 中有一些所谓的预定义变量（Predefined Variable）。每当 MATLAB 启动，这些变量就被自动产生。建议读者在编写指令和程序时，不要对表 2-1 所列预定义变量重新赋值，以免产生混淆。

表 2-1　　MATLAB 的预定义变量

预定义变量	含　义	预定义变量	含　义
ans	计算结果的缺省变量名	nargin	函数输入宗量数目
eps	机器零阈值	nargout	函数输出宗量数目
Inf 或 inf	无穷大，如 1/0	realmax	最大正实数
pi	圆周率 π	realmin	最小正实数
NaN 或 nan	不是一个数，如 0/0		

2.2.2 数据类型

MATLAB 中定义了 6 种数据类型，即字符型（char）、双精度型（double）、稀疏型（sparse）、无符号整型（unit8）、单元型（cell）和结构型（struct），各种数据类型的说明如表 2-2 所示。

表 2-2　　各种数据类型的范例与说明

类　型	范　例	说　明
double	[1.21;0.01]	双精度数组，一般的运算、函数、数组都支持这种类型
char	'ABCD'	字符数组，每个字符用一个 16 位数据来表示
sparse	speye(4)	双精度稀疏数组，稀疏数组中只保存非 0 和其向量。运算必须配合特殊的运算符
cell	{'ab' 1.21}	单元数组，可以包含不同类型的数组
struct	Dog.name='Hali' Dog.age=2 Dog.color=red	结构数组，也可以将不同的数据类型包含在同一个变量名称下，但结构数组另外含有数组名称
unit8	Unit8(56.7)	无符号的 8 位整数，数字范围为 0～255。当前，无数学运算支持它，但是图像的数据都以此类型保存，所以在对图像进行运算之前，需要将其转换为双精度，计算完后，再转换为 unit8 用于图像显示
Userobject		用户自定义的数据类型

2.2.3 基本赋值语句

MATLAB 实际上可以认为是一种解释性语言，读者可以在 MATLAB 工作环境下键入命令，也可以用 MATLAB 语言编写程序，这样 MATLAB 软件对键入的命令或编写的程序进行翻译，然后在 MATLAB 环境下运行处理，最后返回运算结果。

MATLAB 最基本的赋值语句结构为：

变量名=表达式

其中符号左边的变量名列表为 MATLAB 语句的返回值，等号右边的是表达式的定义，也可以是矩阵运算，也可以是 MATLAB 提供的函数调用，表达式可以用分号（;）结束，也可以用换行符结束。分号结束，赋值结果不会在屏幕上显示，否则左边变量的运行结果全部显示。在程序调试期间，显示某些关键参数将有助于算法分析，但在程序运行阶段，大量中间结果显示会降低运行速度。尤其是在检测算法的运行速度时，一定要避免图像矩阵数据的显示。在调用函数时，MATLAB 允许同时返回多个运行结果，这时等号左边是用[ ]括起来的变量列表。

【例 2-1】在矩阵 A=[1 4 7 3]中找出最大值 Max 及其所在位置 n。

```
A=[1,4,7,3] ;[Max, n]=max(A) 回车:
Max =
     7
n =
     3
```

2.2.4 工作空间的管理

MATLAB 的工作空间包含了 MATLAB 程序中使用的变量表，可以用 who 或 whos 命令来查看，who 命令只返回一个简单的变量列表，whos 命令返回全部变量的变量名（Name）、大小（Size）、元素数（Element）、字节数（Bytes）、表现密度（Density）、有无复数（Complex），除了对单个变量给出相应的信息外，还将给出整个变量空间的占用情况。

用户了解了当前工作空间中的现有变量名后，可以调用 clear 命令来删除其中的一些不再使用的变量，这样可以使得整个工作空间更简洁，例如用户想删除工作空间中的 I 和 J 变量，则可以使用下面的 MATLAB 命令。

```
clear I J
```

注意：在这一命令下 I 与 J 之间不能加逗号，否则该命令会被错误地解释成删除 I 变量，然后开始下一个语句（J），而该语句将被解释成将 J 变量的内容显示出来，这样 J 变量将不会被删除，如果用户想删除整个工作空间中的所有变量，则可以使用 clear 命令，在该命令后不加任何参数。

此外，clear 命令还可以用来删除工作空间中的全局变量和编译结果，使用 clear 命令时还应该注意，它所带的参数必须是工作空间中现有的变量名，否则将给出错误信息，并终止程序运行。

如果用户想查询在当前的工作空间中是否存在一个变量时，可以使用 exist 命令来实现，该函数的调用格式为：

```
i=exist('A');
```

返回值 i 表示变量名存在的形式，含义如下：

i=1 表示在当前工作空间中存在一个变量名为 A 的矩阵；

i=2 表示在 MATLAB 的工作路径下存在一个名为 A.m 的文件；

i=3 表示在 MATLAB 的工作路径下存在一个名为 A.mex 的文件；

i=4 表示存在一个编译好的名为 A.m 的 SIMULINK 文件；

i=5 表示存在一个 MATLAB 函数；

i=0 表示不存在和 A 有关的变量和文件。

图像处理时，往往需要先得到图像的大小，可以使用 size()命令来实现，该函数的调用形式为：

```
[m, n]=size(A)
```

其中 A 为矩阵名，m 和 n 分别为矩阵的行数和列数。

如果要测试一个向量的长度时，可以使用 length 函数：

```
n=length(A)
```

n 将返回向量 A 的元素个数。

2.3　MATLAB 运算符

MATLAB 的运算符有下列 3 种类型：

（1）算术运算符——处理两个运算元的数学运算，例如加、减、乘和除等。

（2）关系运算符——比较两个运算元之间的关系，例如大于、小于和等于等。

（3）逻辑运算符——处理两个运算元的逻辑运算，例如 AND 和 OR，返回值为“TRUE”或“FALSE”。

通常在编写程序时，可能会在同个式子内同时用到两种或 3 种不同类型的运算符，这时 MATLAB 规定的处理顺序依次为算术运算符、关系运算符、逻辑运算符。如果需要改变某种运算顺序时，可以采用括号()，括号内的运算优先次序最高。

2.3.1　算术运算符

基本运算包括矩阵四则运算、矩阵的幂、单个数的四则运算和单个数的幂运算。在同一个运算式中，如果有多个运算符存在，那么 MATLAB 会依照规定的优先顺序执行运算。

（1）加法运算（+）

假设在 MATLAB 环境中有两个矩阵 A 和 B，若 A、B 维数相同，则可以由“+”运算符实现矩阵的加法，若 A、B 维数不同，则 MATLAB 将会自动地给出错误信息，提示用户两个矩阵的维数不匹配。

（2）减法运算（－）

减法运算符“－”可以实现两个维数相同的矩阵的减法操作。

（3）矩阵的转置（'）

假设矩阵 A 为一个 $n \times m$ 矩阵，则其转置矩阵后的元素定义为：

$$b_{ji} = a_{ij}, \quad i = 1, \cdots, n, \qquad j = 1, \cdots, m \tag{2-1}$$

（4）乘法运算（×）

设矩阵在 A 为 $n \times m$ 矩阵，B 为 $m \times r$ 矩阵，则 A 与召的乘积 $C = AB$，为 $n \times r$ 矩阵，其各

个元素为

$$C_{ij}=\sum_{k=1}^{m}a_{ik}b_{kj}\,,\quad i=1,\cdots,n\quad j=1,\cdots,r \tag{2-2}$$

（5）除法

① 矩阵左除法（\）

MATLAB 中用运算符“\”来表示矩阵左除，A\B 即由 Gauss 消元法来获得线性方程 $AX=B$ 的解 X。

② 矩阵右除（/）

在 MATLAB 中用“/”运算符表示矩阵右除。A/B 即为 AB^{-1}。

（6）点除法（./）

① MATLAB 中定义运算符（./）为点右除法，A./B 表示 A 和 B 对应元素间的除法，所以 A 和 B 维数必须相同，除非其中之一为常数。

② 左点除法（.\）

A.\B 表示 A 和 B 对应元素间的左除法。

（7）点乘法（.*）

在 MATLAB 中，运算符“.*”表示相同维数矩阵对应元素间的乘法。

（8）幂运算（^）

对于矩阵 A，其幂运算在 MATLAB 中表示为 A^x，其中 x 为幂。当 x 为整数时，幂运算的结果是将 A 矩阵自乘 x 而得出，如果 x 为负数，则将矩阵 A 自乘 x 后对结果进行求逆运算而得出，如果 x 为分数或小数，如 $x=n/m$，其中 n 和 m 均为整数，则首先将 A 矩阵自乘 n 次，再将结果开 m 次方。

（9）点幂运算（.^）

在 MATLAB 中，运算符“.^”表示对每个矩阵元素都进行相应乘方。

（10）共轭复数

MATLAB 中，运算符“'”表示共轭复数运算。

2.3.2 关系运算符

关系运算符通常结合 if，while，for，switch 等运用于程序流程控制上。MATLAB 的关系运算有：

（1）相等（==）对维数相同的矩阵 A 和 B，A==B 执行 A，B 两矩阵相对应元素的比较，如果相等则在相应元素处返回 1，如不相等返回 0；

（2）小于（<）；

（3）小于等于（<=）；

（4）大于（>）；

（5）大于等于（>=）；

（6）不等于（~=）。

2.3.3 逻辑运算符

MATLAB的逻辑运算有4种，即AND、OR、NOT和XOR。与关系运算符类似，逻辑运算符用来处理两个运算元，这两个运算元必须是相同大小的数组或向量，其中一个也可以是常数。如果两个运算元是同样大小的数组，那么运算符会处理两个运算元中相对元素的数值运算。如果其中一个是常数，则MATLAB会将这个常数跟运算元的每一个元素做运算。运算结果为1表示TRUE，0表示FALSE。值得注意的是，在处理逻辑运算时，运算元只有两个值，即0或1。所以，如果指定的数为0，MATLAB认为其为0，而任何数不等于0则认为是1。

（1）与运算（&）

如果有两个维数相同的矩阵*A*和*B*，*A*&*B*表示对它们进行与运算，如果*A*和*B*对应元素均不为0，则相应结果为1，否则输出为0。

（2）或运算（|）

对两维数相同的矩阵*A*、*B*，*A*|*B*表示对它们进行或运算，当*A*和*B*的相应元素至少有一个不为零时，输出为1，当*A*、*B*两对应元素均为0时，输出为0。

（3）非运算（~）

~*A*表示对矩阵*A*进行非运算，即如果*A*中对应元素为0，则相应输出为1，如果对应元素为非零，则相应输出为0。

（4）异或运算（XOR）

XOR(*A*,*B*)表示对*A*、*B*进行异或运算，当*A*和*B*相应的元素相同（均为0或1），相应输出为1，如果*A*、*B*中不相同或只有一个不为0时，则输出为0。

除以上的基本逻辑运算以外，MATLAB还提供了一些逻辑函数。

（5）ALL

ALL(*A*)检查*A*向量中的元素是否全为真，如果所有元素均为非0，则返回1，否则返回1；对矩阵而言，对矩阵的第*x*列进行检查，返回元素为1或0的行向量。

（6）ANY

ANY(*A*)检查*A*向量中是否有非0元素，如果有非0元素，则返回1，否则返回0；对矩阵而言，对矩阵的第*x*列进行检查，返回元素为1或0的行向量。

（7）ISISINTIE

检查元素是否为有限值，如果元素是有限值，则相应输出为1，如为无限值，则相应输出为0。

（8）ISNAN

检查元素是否为NAN，如果是则相应输出为1，如不是则相应输出为0。

（9）ISINF

ISINF(*A*)检查*A*向量中的元素是否为无限值，如果是，则相应输出为1，否则相应输出结果为0。

2.4 控制语句

和其他高级语言一样，MATLAB也提供了条件转移语句、循环语句等一些常用的控制语

句，从而使得 MATLAB 语言的编程显得十分灵活。MATLAB 支持的控制语句和 C 语言中的控制语句格式是很相似的。下面将介绍一些常用的 MATLAB 语言控制语句及其使用方法与实例。

2.4.1 循环控制语句

MATLAB 中可以使用两种循环语句：for 语句和 while 语句。这两种语句的基本格式和 C 语言中的循环语句很相似，例如 for 语句的基本格式为：

```
for 循环变量：表达式 1：表达式 3：表达式 2
循环语句组
end
```

注意：这里的循环语句是以 end 结尾的，这和 C 语言的结构不完全一致。在 C 语言循环中，循环体的内容是以大括号{}括起来的，而在 MATLAB 语言中，循环体的内容是以循环语句和 end 语句括起来的，所以在使用 MATLAB 时应注意这一点。

在 MATLAB 的循环语句基本格式中，循环变量可以取作任何 MATLAB 变量，表达式 1，表达式 2 和表达式 3 的定义和 C 语言相似，即首先将循环变量的初值赋成表达式 1 的值，然后再求取表达式 2 的值，如果此时循环变量的值介于表达式 1 和表达式 2 的值之间，则执行循环变量自增一个 s3 的值，然后再判断循环变量是否介于表达式 1 和表达式 2 之间，如果满足仍再执行循环体，直至不满足为止，这时将结束循环语句的执行，而继续执行后面的语句。

【例 2-2】求 10!的值，可以用下面的程序：

```
result=1;
for i=2:1:10
    result=result*i;
end
```

在上面的式子中，可以看到 for 循环语句中表达式 3 的值为 1。在 MATLAB 实际编程中，如果表达式 3 的值为 1，则可以在该句中省略，故该句可以简化成 for i=l:10。

MATLAB 语言提供了另一种循环语句结构 while，该循环语句的结构为：

```
while （条件式）
循环体语句组：
end
```

其执行方式为，若条件式中的条件成立，则执行循环体的内容执行后再判断表达式是否仍然成立，如果表达式不成立则跳出循环，向下继续执行。

重新考虑上面的程序，如果改用 while 循环结构，则可以写出下面的程序

```
result=1;
i=2;
while(i<=10)
      result=result*i;
      i=i+1;
end
```

MATLAB 提供的循环结构 for 和 while 是允许多级嵌套的，而且它们之间也允许相互嵌套。

2.4.2　条件转移语句

除了前面介绍的循环语句结构之外，MATLAB 还提供了各种条件转移语句的结构，使得 MATLAB 语言更易于使用。MATLAB 提供的条件语句最简单的格式是由关键词 if 引导的，其格式为

```
if (条件式)
条件块语句组
end
```

当给出的条件式成立时，则执行该条件块结构中的语句内容，执行完之后继续向下执行，若条件不成立，则跳出条件块而直接向下执行。

【例 2-3】如果将例 2-2 中给出的问题变成求出满足 $r!>1000$ 的最小 r，读者可以针对这一问题编写如下的程序段。

```
r=1;
i=2;
for i=2:1000
    if (r<1000)
        r=r*i;
    else
        break;
    end
end
```

注意：这里使用了 break 命令，其作用就是终止上一级的 for 语句循环过程。

while 循环结构在 MATLAB 下也起着相当重要的作用，因为在 MATLAB 下没有提供绝对转移的指令，所以这样的功能就必须通过像 while 那样的循环结构来实现，当然，这样的实现离不开和 if 型条件语句的配合。

MATLAB 还提供了其他两种条件结构，if-else 和 if-else if，这两种格式的调用方法分别为：

```
if (条件式)
条件块语句组 1
else
条件块语句组 2
end
```

其应用可参考例 2-3。

```
if (条件式)
条件块语句组 1
else if (条件式)
……
end
```

这些语句的结构和功能与 C 等程序设计语言是相同的。

2.4.3　开关控制语句

另有一种流程控制的方法是 switch。如果在一个程序中，必须针对某个变量值来做多种不

同的执行，则 switch 比 if、else 更为方便。另一方面，合理使用 switch 语句也可以使程序更具有可读性。switch 的语法如下：

```
switch 分支条件(数值或字符串)
case 数值(或字符串)条件一
运算指令一
case 数值(或字符串)条件二
运算指令二
otherwise
运算指令 N
end
```

基本的 switch 语句包含下列元素。

switch，switch 语句的开始，紧接着分支条件。分支条件可以是一个函数、变量或者表达式。

case，依照分支条件的值，不同的 case 可以定义不同的运算指令。而紧接在 case 后面的就是此 case 的分支条件，之后接着一个或一串运算指令。

otherwise，若不符合所有 case 的条件，则程序就会执行 otherwise 下面的表达式。

end，switch 语句的结束。

switch 实际上也是利用分支条件(数值条件)比较来实现其目的。如果上面的判断为 TRUE，则符合条件，并且执行紧接的运算。反之，如果返回值为 FALSE，则继续检验下一个 case，直到最后一个失败，才执行 otherwise 下面的程序片段。

与 C 语言相比，MATLAB 的 switch 语句中缺少了 break。因为在 C 语言中，程序在检验某个 case 符合并执行相应的运算后，还继续检验下一个 case，直到全部检验完。所以，加入 break，让程序只运算第一个条件符合的运算。MATLAB 则只运行第一个检验符合条件的 case。

MATLAB 还有一些其他非常有用的控制程序的指令，见表 2-3。

表 2-3　控制程序流的其他常用指令

指令及使用格式	使 用 说 明
v=input('message') v=input('message', 's')	指令执行时，“控制权”交给键盘；待输入结束，按下 Enter 键，“控制权”交还 MATLAB，message 是提示用的字符串。第一种格式用于键入数值、字符串、元胞数组等数据；第二种格式，不管键入什么，总以字符串形式赋给变量 v
keyboard	遇到 keyboard 时，将“控制权”交给键盘，用户可以从键盘输入各种 MATLAB 指令。仅当用户输入 return 指令后，“控制权”才交还给程序，它与 input 的区别它允许输入任意多个 MATLAB 指令，而 input 只能输入赋给变量的“值”
continue	跳过位于其后的循环中的其他指令，执行循环的下一个迭代
pause pause(n)	一种格式使程序暂停执行，等待用户按任意键继续；第二种格式使程序暂停 *n* 秒后，再继续执行
return	结束 return 指令所在函数的执行，而把控制转至主调函数或者指令窗。否则，只有待整个被调函数执行完后，才会转出

续表

指令及使用格式	使 用 说 明
error('message')	显示出错信息 message，终止程序
lasterr	显示出错信息 message，终止程序
lastwarn	显示 MATLAB 自动给出的最新警告程序继续运行
warning('message')	显示警告信息 message，程序继续运行

2.5　M 脚本文件和函数文件

对于一些比较简单的问题，从指令窗中直接输入指令进行计算是十分轻松简单的事。但随指令数的增加或随控制流复杂度的增加，以及重复计算要求的提出，直接从指令窗进行计算就显得烦琐。此时，脚本文件最为适宜。“脚本”本身反映这样一个事实：MATLAB 只是按文件所写的指令执行。这种文件的构成比较简单，其特点是：

（1）它只是一串按用户意图排列而成的（包括控制流向指令在内的）MATLAB 指令集合；

（2）脚本文件运行后，所产生的所有变量都驻留在 MATLAB 基本工作空间（Base workspace）中。只要用户不使用 clear 指令加以清除，且 MATLAB 指令窗不关闭，这些变量将一直保存在基本工作空间中。基本空间随 MATLAB 的启动而产生；只有关闭 MATLAB 时，该基本空间才被删除。

2.5.1　M 文本编辑器

MATLAB Editor/Debugger 是一个集编辑与调试两种功能于一体的工具环境。

（1）为创建新 M 文件，启动编译器的 3 种操作方法。

① MATLAB 指令窗运行指令 edit。

② MATLAB 指令窗工具条上的 D 图标。

③ MATLAB 指令窗的“File→New”菜单，再从右拉菜单中选择“M-file”项。

（2）打开已有的 M 文件的 3 种操作方法。

① MATLAB 指令窗运行指令 edit filename。filename 是待打开文件名，可不带扩展名。

② MATLAB 指令窗工具条上的图标，再从弹出对话框中点选所需打开的文件。

③ MATLAB 指令窗[File:Open]子菜单，再从弹出对话框中点选所需打开的文件。

（3）经编写或修改后，文件的保存方法。

单击编辑器工具条上的图标，也可选取编辑器的[File：Save]子菜单。若是已有文件，则便完成了保存；若是新文件，则会弹出“保存”文件对话框，经过存放目录和文件名的选择，才可完成保存。

2.5.2 M 函数文件

与脚本文件不同，函数文件（Function File）犹如一个“黑箱”。从外界只能看到传给它的输入量和送出来的计算结果，而内部运作是藏而不见的。它的特点如下。

（1）从形式上看，与脚本文件不同，函数文件的第一行总是以“function”引导的“函数声明行”（Function Declaration Line）。该行还罗列出函数与外界的联系的全部“标称”输入、输出宗量。但对“输入输出宗量”的标称数目并没有限制，即可以完全没有输入、输出宗量，也可以是任意数目。

（2）MATLAB 允许使用比“标称数目”较少的输入、输出宗量，实现对函数的调用。

（3）从运行上看，与脚本文件运行不同，每当函数文件运行，MATLAB 就会专门为它开辟一个临时工作空间，称之谓函数工作空间（Function Workspace）。所有中间变量都存放在函数工作空间中。当执行完文件最后一条指令或遇到 return 时，就结束该函数文件的运行，同时该临时函数空间及其所有的中间变量就立即被清除。

（4）函数空间随着具体的 M 函数文件的被调用而产生，随调用结束而删除。函数空间是相对基本空间独立的、临时的。在 MATLAB 运行期间，可以产生任意多个临时函数空间。

（5）假如在函数文件中，发生对某脚本文件的调用，那么该脚本文件运行产生的所有变量都存放于该函数空间之中，而不是存放在基本空间。

2.5.3 文件的一般结构

由于从结构上看，脚本文件只是比函数文件少一个“函数声明行”，所以只需描绘清楚函数文件的结构，脚本文件的结构也就无需过多说明。典型 M 函数文件的结构如下。

（1）函数声明行（Function Declaration Line）：位于函数文件的首行，以 MATLAB 关键字 function 开头，函数名以及函数的输入、输出宗量都在这一行定义。

（2）H1 行（The First Help Text Line）：紧随函数声明行之后以%开头的第一注释行。按 MATIAB 自身文件的规则，H1 行包含：大写体的函数文件名；运用关键词简要描述的函数功能。该 H1 行供 look for 关键词查询和 help 在线帮助使用。

（3）在线帮助文本（Help Text）区：H1 行及其之后的连续以%开头的所有注释行构成整个在线帮助文本。它通常包括：函数输入、输出宗量的含义，调用格式说明。

（4）编写和修改记录：与在线帮助文本区相隔一个“空”行，也以%开头，标志编写及修改该 M 文件的作者和日期、版本记录。它用于软件档案管理。

（5）函数体（Function Body）：为清晰起见，它与前面的注释以“空”行相隔。这部分内容由实现该 M 函数文件功能的 MATLAB 指令组成。它接收输入宗量，进行程序流控制，得到输出宗量。其中为阅读、理解方便，也配置适当的空行和注释。若仅从运算角度看，惟有“函数声明行”和“函数体”两部分是构成 M 函数文件所必不可少的。

【例 2-4】M 函数文件示例（detectedge.m）

```
function [RI]=detectedge(I)
% 用数学形态学的腐蚀算法对二值图像 I 检测边缘
% I 输入的二值化图像
% RI 输出的边缘图像
```

```
% 编写于 2002 年 10 月， 第一次修改于 2003 年 8 月

I=1-I;
se1=strel('square',3);
I1=imerode(I,se1);
RI=I-I1;
```

2.6　MATLAB 图像处理初步

MATLAB 是一种基于向量（数组）而不是标量的高级程序语言，因而 MATLAB 从本质上就提供了对图像的支持。从图像的数字化过程知道，数字图像实际上就是一组有序的离散数据，使用 MATLAB 可以对这些离散数据形成的矩阵进行一次性的处理。较其他标量语言而言，这是非常有优势的一点。为了使读者对 MATLAB 图像产生一个整体概念，下面给出两个图像处理的实例，详细介绍每一步的操作和使用的命令，帮助读者掌握如何使用 MATLAB 图像处理的函数和相关操作等。例子中涉及到的图像处理技术读者可以在后续章节进行学习和掌握。

在启动 MATLAB 运行本书的所有例子之前，必须保证图像处理工具箱已经安装。例 2-5 图像处理的基本操作介绍如何在 MATLAB 环境下读取图像，显示图像，简单的灰度处理及图像的保存等。例 2-6 高级图像处理的操作介绍如何在 MATLAB 环境下对图像进行背景估计、图像相减、二值化、图像中的目标的属性统计等图像操作。

2.6.1　图像处理基本操作

【例 2-5】展示了图像处理的基本操作

1. 读入并显示一幅图像

首先，清除 MATLAB 所有的工作平台变量，关闭已打开的图形窗口：

```
clear;
close all;
```

然后使用图像读取函数 imread 函数来读取一幅图像。假设要读取的图像 cameraman.tif（该图像是图像处理工具箱自带的图像，如果读者向读取自己的图像，则 Current Directory 栏中设置为自己图像所在目录即可），并将它存储在一个名为 I 的数组中：

```
I=imread('cameraman.tif');
```

使用 imshow 函数来显示图像 I：

```
imshow(I);
```

显示结果如图 2-2 所示。



图 2-2 图像 cameraman.tif 的显示效果

2. 检查内存在中的图像

使用 whos 函数来查看图像数据 I 是如何存储在内存中的：

```
whos
```

MATLAB 将做出如下的响应：

```
Name      Size                Bytes  Class
I      256x256                65536  uint8 array
Grand total is 65536 elements using 65536 bytes
```

3. 改变图像的大小

MATLAB 使用 imresize 函数来改变图像的大小。将原来的图像缩小为原来的一半：

```
I2=imresize(I,0.5);
```

使用 figure 命令创建一个新的窗口，从而避免新图像显示时，会覆盖原来的图像。

```
figure,imshow(I2);
```

调整大小后的图像，如图 2-3 所示。



图 2-3 缩小为原来一半的图像

4. 保存图像

将新的调整后的图像 I2 保存到磁盘中。假设希望将该文件保存为 BMP 格式的图像文件，使用 imwrite 函数并指定一个文件名，该文件的扩展名为.bmp：

```
imwrite(I2,'cameraman.bmp');
```

5. 检查新生成的文件内容

使用 imfinfo 函数来观察上述语句写了什么内容到磁盘上。为了保证 MATLAB 在屏幕上能显示图像输出结果，imfinfo 命令后不加分号：

```
imfinfo('cameraman.bmp')
```

MATLAB 屏幕显示如下：

```
Filename: 'cameraman.bmp'
FileModDate: '25-Nov-2003 15:29:58'
FileSize: 17462
Format: 'bmp'
……
```

显示了图像的文件名、保存日期、文件大小和图像文件的格式等。

2.6.2　高级图像处理初步

【例 2-6】展示了高级图像处理技术。

在这个练习中，主要是对一幅灰度图像 rice.tif 进行一些较为高级的操作。图像处理的目的是消除 rice.tif 图像中的亮度不一致的背景，并使用阈值分割将修改后的图像转换为二值图像，使用成员标记返回图像中目标对象的个数以及统计特性。

1. 读取和显示图像

清除 MATLAB 工作空间的所有变量，关闭已打开的图形窗口，读取显示灰度图像 rice.tif：

```
clear;
close all;
I=imread('rice.tif');
Imshow(I);
```

显示结果如图 2-4 所示。

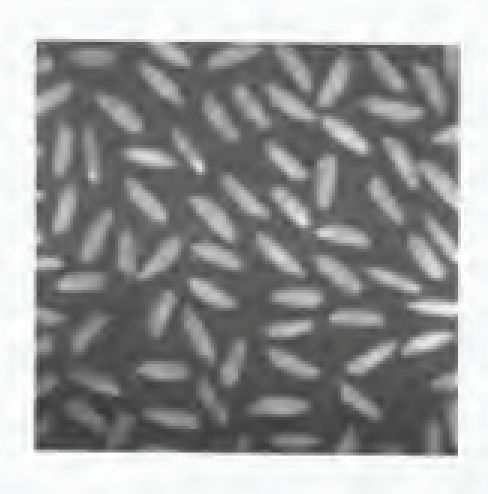


图 2-4　rice.tif 的灰度图像

2. 估计图像背景

观察 rice.tif 图像发现中心位置的背景亮度要高于其他部分的亮度。创建一个半径为 16 像素的圆盘形结构，用该结构对 rice.tif 图像进行数学形态学的开操作后，估计背景亮度：

```
Stru=strel('disk',16);
Back=imopen(I,Stru );
```

3. 从原始图像中减去背景图像

将 Back 从原始图像中减去，得到一个背景比较一致的图像，如图 2-5 所示。

```
I2=imsubtract(I,Back);
figure, imshow(I2);
```



图 2-5　去除背景后的 rice.tif 图像

4. 使用阈值操作将图像转换为二值图像

首先，利用 graythresh 函数得到图像的全局阈值，然后使用 im2bw 将灰度图像转换为二值图像，如图 2-6 所示。

```
Threshold=graythresh(I2);
BW=im2bw(I2,Threshold);
figure, imshow(BW);
```



图 2-6　rice.tif 图像阈值分割后的二值图像

5. 检查图像中的目标对象数量

使用 bwlabel 函数可以确定 rice.tif 二值化图像中的米粒个数：

```
[Labeled Number_Object]=bwlabel(BW,8);
Number_Object=80
```

bwlabel 函数标识二值图像 BW 中的所有相关成分，所以图 2-6 中部分相互连接的米粒，该函数将它们视为同一个对象。

6. 计算图像中对象的统计属性

使用 regionprops 函数来获取图像中的目标对象属性，并将属性保存到一个结构体数组中。采用 max 和 mean 可以得到最大米粒的大小和米粒的平均大小。

```
GrainData=regionprops(Labeled, 'basic');
  AllGrain=[GrainData.Area];
max(AllGrain)
```

MATLAB 屏幕显示数据：

```
ans =
      698
```

获取米粒的平均大小：

```
mean(AllGrain)
ans =
  251.0750
```

2.7 图像格式与 MATLAB 图像类型

2.7.1 常用图像格式

在介绍图像格式之前，首先要介绍有关图像格式的一个重要概念：调色板。调色板是包含不同颜色的颜色表，每种颜色以红、绿、蓝 3 种颜色的组合来表示，图像的每一个像素对应一个数字，而该数字对应调色板中的一种颜色，如某像素值为 1，则表示该颜色为调色板的编号为 1 的颜色。调色板的单元个数是与图像的颜色数相对应的，256 色图像的调色板就有 256 个单元。真彩图像的每个像素直接用 R、G、B 量值来表示颜色，因此不需要调色板。注意，对于 16 色或 256 色图像并非全部的图像都采用相同的 16 种或 256 种颜色，由于调色板中定义的颜色不同，则不同图像用到的颜色是千差万别的，所谓 16 色或 256 色图像，只是表示该幅图像最多只能有 16 种颜色或 256 种颜色。不同的图像有不同的调色板，在多媒体系统中如果需要同时显示多幅图像时，由于系统在同一时刻只能支持有限的颜色，如使用 256 色驱动程序，则系统以 256 色来同时显示多幅图像，因此，必须使用调色板编辑工具进行调整，以达到不失真的同时显示多幅图像的效果。

图像格式指的是存储图像采用的文件格式。不同的操作系统、不同的图像处理软件，所支持的图像格式都有可能不同。在实际应用中经常会见到以下几种图像格式。

BMP（Bitmap）文件：BMP 文件是 Microsoft Windows 所定义的图像格式，最早应用在 Microsoft 公司的 Microsoft Windows 窗口系统中。BMP 图像文件文件有这样一些特点：该结构只能存放一幅图像；只能存储 4 种图像数据：单色、16 色、256 色、真彩色；图像数据有压缩或不压缩两种处理方式；调色板的数据存储结构较为特殊，其存储格式不是固定的，而是与文件头的某些具体参数（如像素位、压缩算法等）密切相关的。Windows 设计了两种压缩方式：RLE4 和 RLE8。RLE4 只能处理 16 色图像数据；而 RLE8 则只能压缩 256 色图像数据。BMP 图像文件的文件结构可分为 3 部分：表头、调色板和图像数据。表头长度固定为 54 个字节。只有真彩色 BMP 图像文件内没有调色板数据，其余不超过 256 种颜色的图像文件都必须设定调色板信息。

GIF（Graphics Interchange Format）文件：GIF 文件是由 CompuServe 公司为了方便网络用户传送图像数据而制定的一种图像文件格式。GIF 图像文件经常用于网页的动画、透明等特技制作。GIF 文件有这样一些特点：文件具有多元化结构，能够存储多张图像；调色板数据有通用调色板和局部调色板之分；采用优于 RLE 压缩法的改进版——LZW 压缩法；图像数据一个字节存储一点；文件内的各种图像数据区和补充区多数没有固定的数据长度和存放位置，为了方便程序寻找数据区，就以数据区的第一个字节作为标识符，以使程序能够判断所读到的是哪种数据区；图像数据有顺序排列和交叉排列两种排列方式；图像最多只能存储 256 色图像。GIF 图像文件结构一般由 7 个数据单元组成，它们分别是：表头、通用调色板、图像数据区以及 4 个补充区。表头和图像数据区是文件不可缺少的单元，通用调色板和其余的 4 个补充区是可选内容。GIF 图像文件可以有多个图像数据区，而每个图像数据区存储一幅图像，通过软件处理和控制使这些分离的图像形成连续有动感的图示效果。

TIFF（Tag Image File Format）文件：TIFF 文件是由 Aldus 公司与微软公司共同开发设计的图像文件格式。它有这样一些特点：善于应用指针的功能，可以存储多幅图像；文件内数据区没有固定的排列顺序，但规定表头必须在文件前端，标识信息区和图像数据区在文件中可以随意存放；可制定私人用的标识信息；除了一般图像处理常用的 RGB 模式之外，TIFF 图像文件还能够接受 CMYK、YCbCr 等多种不同的图像模式；可存储多份调色板数据；调色板的数据类型和排列顺序较为特殊；能提供多种不同的压缩数据的方法，以方便使用者选择；图像数据可分割成几个部分进行分别存档。TIFF 图像文件主要由 3 部分组成：表头、标识信息区和图像数据区。文件内固定只有一个表头，且一定位于文件前端。表头有一个标志参数指出标识信息区在文件中的存储地址，标识信息区有多组标识信息用于存储图像数据区的地址。每组标识信息长度固定为 12 个字节，前 8 个字节分别代表标识信息的代号（两个字节）、数据类型（两个字节）、数据量（4 个字节），最后 4 个字节则存储数据值或标志参数。文件末尾有时还存放一些标识信息区容纳不下的数据，例如，调色板数据就是其中的一项。

PCX 文件：PCX 文件是由 Zsoft 公司在 20 世纪 80 年代初期设计的，专用于存储该公司开发的 PC Paintbrush 绘图软件所生成的图像画面数据。目前 PCX 文件已成为 PC 机上较为流行的图像文件。PCX 图像文件具有这样一些特点：一个 PCX 图像文件只能存放一张图像画面；使用 RLE 压缩方法进行数据压缩；PCX 图像文件有多个版本，能处理多种不同模式下的图像数据；4 色和 16 色 PCX 图像文件有可设定或不设定调色板数据的两种选项；16 色图像数据可由 1 个或 4 个 bit Plane（颜色的 RGB 等级）来处理。

JPEG 格式：它是由 Joint Photographic Experts Group 制定的图像压缩格式，其正式名称为“连续色调静态图像的数字压缩和编码”，是一种通用的静态图像压缩编码标准，可以用不同的压缩比例对文件格式进行压缩。JPEG 压缩技术十分先进，它采用最少的磁盘空间来得到较好的图像质量。

PSD 格式：这是 Adobe 公司开发的图像处理软件 Photoshop 中自建的标准图像文件格式，由于 Photoshop 软件被广泛地应用，因而这种格式也很流行。

PCD 格式：这是 KODAK 公司所开发的 Photo CD 专用存储格式，由于其文件特别大，所以常存储在 CD-ROM 上，但其应用领域特别广。

WMF 矢量格式：这是微软公司开发的矢量图形格式，在 Office 等软件中得到了大量的应用。

2.7.2 MATLAB 图像类型

图像类型是指数组数值与像素颜色之间定义的关系，注意其与图像格式概念的区别。对图像进行处理，首先要了解图像的保存的类型。MATLAB 图像处理工具箱支持 5 种类型的图像，下面将一一介绍。

MATLAB 使用 3 种存储格式来存储图像：uint8（8 位无符号整数）、uintl6（16 位无符号整数）和双精度。在本书中未特别说明的函数都可以对任意一种存储类型进行操作。

1. 二进制图像

在一幅二进制图像中，每一个像素将取两个离散数值（0 或 1）中的一个，从本质上说，这两个数值分别代表状态"开"（on）或"关"（off）。其中 1 表示白色，0 表示黑色。

二进制图像能够使用 uint8 或双精度类型的数组来存储（图像处理工具箱不支持 uintl6 类型的二进制图像）。uint8 类型的数组通常比双精度类型的数组性能更好，因为 uint8 数组使用的内存要小得多。在图像处理工具箱中，任何返回一幅二进制图像的函数都使用 uint8 逻辑数组存储该图像。工具箱使用一个逻辑标志来指示 uint8 逻辑数组的数据范围。如果逻辑状态为"开"（on），那么数据范围是[0,1]；如果为"关"（off），那么工具箱假定数据范围为[0,255]。图 2-7 给出了一幅典型的二进制图像。

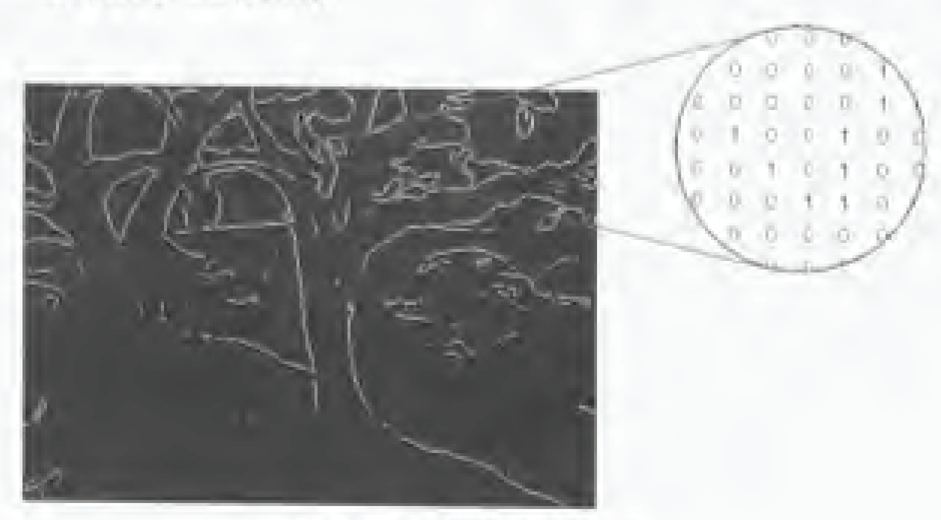


图 2-7 典型二进制图像示例

2. 索引图像

索引图像是一种把像素值直接作为 RGB 调色板下标的图像。

在 MATLAB 中，一幅索引图像包含一个数据矩阵 X 和一个调色板矩阵 map，数据矩阵可以是 uint8、uintl6 或双精度类型的，而调色板矩阵则总是一个 $m\times3$ 的双精度类型矩阵（其中，m 表示颜色数目），该矩阵的元素值都是[0,1]范围内的浮点数。map 矩阵的每一行指定一个颜色的红、绿、蓝颜色分量。索引图像可把像素值直接映射为调色板数值，每一个像素的颜色通过使用 X 的数值作为 map 的下标来获得：数值 1 表示 map 的第一行，数值 2 表示 map 的第二行，依次类推。

调色板通常与索引图像存储在一起，装载图像时，调色板将和图像数据一同自动装载。图 2-8 说明了一幅索引图像的结构，图中的像素由整数表示，这个整数将作为存储在调色板中的颜色数据的指针。

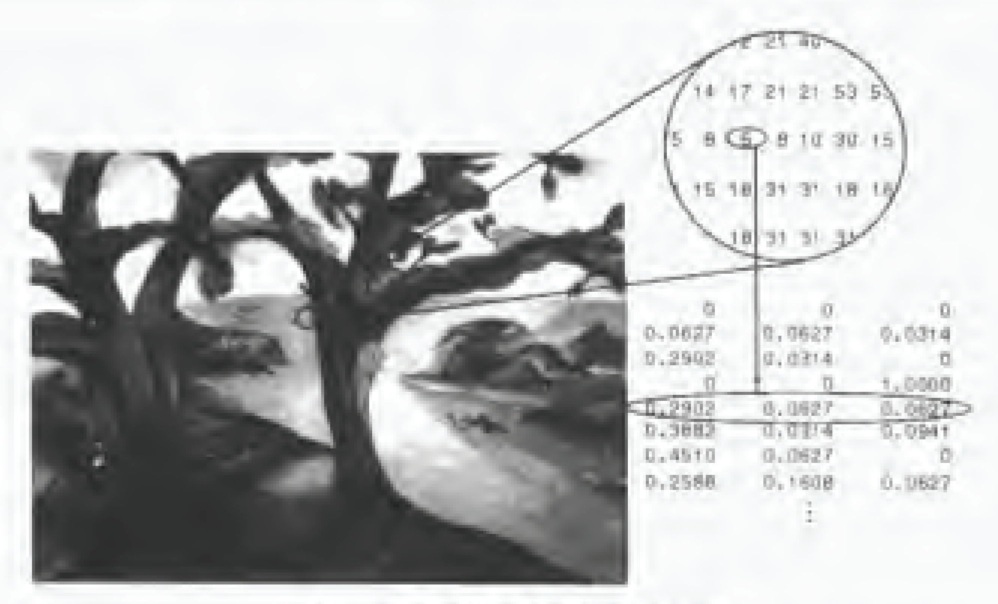


图 2-8　索引图像调色板与数据示例

图像矩阵中的数值与调色板的关系依赖于图像矩阵的类型：如果图像矩阵是双精度类型的，那么数值 1 将指向调色板的第一行，数值 2 将指向调色板的第二行，以此类推；如果图像矩阵是 uint8 或 uintl6 类型的，那么将产生一个偏移量——数值。表示调色板的第一行，数值 1 表示调色板的第二行，以此类推。在以上的图像中，图像矩阵是双精度类型的，因而没有偏移量，数值 5 就代表调色板的第 5 行。

图像处理工具箱对 uintl6 类型索引图像的支持是有限制的，它可以将 uintl6 类型的图像读入 MATLAB 并显示，但是在处理该图像之前必须首先将它转换为 uint8 或双精度类型。如果希望转换为双精度类型，调用 im2double 函数；如果希望将图像转换为 256 色（或更少的颜色），则调用 imapprox 函数。

3. 灰度图像



图 2-9　典型的双精度灰度图像

灰度图像是包含灰度级（亮度）的图像。在 MATLAB 中，灰度图像由一个 uint8、uintl6 或一个双精度类型的数组来描述。

灰度图像实际上是一个数据矩阵 *I*，该矩阵的每一个元素对应于图像的一个像素点，元素的数值代表一定范围内的灰度级，通常 0 代表黑色、255（针对不同存储类型）代表白色。矩阵 *I* 可以是双精度：uint8 或 uintl6 类型的。由于灰度图像存储时不使用调色板因而 MATLAB 将使用一个默认的系统调色板来显示图像。图 2-9 描述了一幅典型的双精度灰度图像。

4. 多帧图像

多帧图像也称为多页图像，是一种包含多幅图像或帧的图像文件。在 MATLAB 内存中，多帧图像是一个四维数组，第四维用来指定帧的序号。多帧图像主要用于需要对时间或场景上相关图像集合进行操作的场合，例如，磁谐振图像切片或电影帧等。

MATLAB 图像处理工具箱提供在同一个数组中存储多幅图像的支持，每一幅单独的图像称为一帧。如果一个数组包含多帧，那么这些图像在第四维中是相联系的。例如，一个包括 5 幅 400×300 大小的 RGB 图像是一个 400×300×3×5 的数组，而相同帧数的灰度或索引图像将是一个 400×300×1×5 的数组。

在一个多帧图像数组中，每一幅图像必须有相同的大小和颜色分量。在多帧索引图像中，每一幅图像还要使用相同的调色板。另外，图像处理工具箱中的许多函数（例如 imshow）只能够对多帧图像矩阵的前两维或前三维进行操作，用户也可以对四维数组使用这些函数，但是必须单独处理每一帧。如果用户将一个数组传递给一个函数，并且数组的维数超过该函数的设计操作维数，那么得到的结果是不可预知的：有些情况下，函数将简单地处理数组的第一帧，其他情况下，函数操作将不会产生任何有意义的结果。

5. RGB 图像

RGB 图像也称为真彩图像，其每一个像素由 3 个数值来指定红、绿和蓝颜色分量。

在 MATLAB 中，一幅 RGB 图像由一个 uint8、uintl6 或双精度类型的 $m \times n \times 3$ 数组（通常称为 RGB 数组）来描述，其中，*m* 和 *n* 分别表示图像的宽度和高度。RGB 图像不使用调色板。每一个像素的颜色由存储在相应位置的红、绿、蓝颜色分量共同决定。RGB 图像是 24 位图像，红、绿、蓝分量分别占用 8 位，因而图像理论上可以包含 2^{24} 种不同的颜色，由于这种颜色精度能够再现图像的真实色彩，所以称 RGB 图像为真彩图像。

在一个双精度类型的 RGB 数组中，每一个颜色分量都是一个[0,1]范围内的数值，颜色分量为(0,0,0)的像素将显示为黑色，颜色分量为(1,1,1)的像素将显示为白色。每一个像素的 3 个颜色分量都存储在数据数组的第三维中。例如，像素(10,5)的红、绿、蓝颜色分量分别存储在 RGB(10,5,1)、RGB(10,5,2)和 RGB(10,5,3)中。

图 2-10 给出了一幅典型的双精度 RGB 图像。为了确定位于(2,3)的像素的颜色，需要察看一组数据 RGB(2,3,1:3)（“:”操作符可以用来直接引用地址从 1～3 的向量数据）。

假设(2,3,1)数值为 0.5176，(2,3,2)数值为 0.1608，(2,3,3)数值为 0.0627，那么像素(2,3)的颜色为(0.5176,0.1608,0.0627)。



图 2-10 典型的双精度 RGB 图像示例

2.7.3 图像类型转换

对于某些操作来说，将一幅图像转换为另一种图像类型有时非常有用。例如，如果用户希望对一幅存储为索引图像的彩色图像进行滤波，那么应该首先将该图像转换为 RGB 格式，此时再对 RGB 图像使用滤波器时，MATLAB 将恰当地滤掉图像中的部分灰度值。如果用户试图对一幅索引图像进行滤波，那么 MATLAB 只能简单地对索引图像矩阵的下标进行滤波，这样得到的结果将是毫无意义的。

当用户将一幅图像从一种类型转换为另一种类型时，得到的图像结果可能与原图像效果会有所不同。例如，如果用户将一幅彩色索引图像转换为灰度图像，得到的结果将是一幅灰度级图像，而并不是一幅彩色图像。

表 2-4 给出了图像处理工具箱中所有的图像类型转换函数。这些函数具有类似的调用格式：函数的输入参数是图像数据矩阵（如果是索引图像，那么输入参数还包括调色板），返回值是转换后的图像（包括索引图像的调色板）。例如，B[IND,COL]＝rgb2ind(RGB)。只有函数 im2bw 的调用格式略有不同，其输入参数还包括一个截取阈值，超过这个阈值的像素被截取为 1，否则为 0。

下面以函数 rgb2gray 以为例说明这些函数的使用方法。

【例 2-7】以下代码的功能是根据图 2-11 所示的 RGB 图像来创建一幅灰度图像（显示结果如图 2-12 所示）：

```
I=imread('Plane211.jpg');
I2=rgb2gray(I);
imshow(I);
figure,imshow(I2);
```





图2-11 原始RGB图像



图2-12 创建的灰度图像

表2-4 MATLAB图像类型转换函数及其功能

函 数	功 能
dither	使用抖动方法，根据灰度图像创建二进制图像或根据RGB图像创建索引图像
gray2ind	根据一幅灰度图像创建索引图像
grayslice	使用阈值截取方法，根据一幅灰度图像创建索引图像
im2bw	使用阈值截取方法，根据一幅灰度图像、索引图像或RGB图像创建二进制图像
ind2gray	根据一幅索引图像创建一幅灰度图像
ind2rgb	根据一幅索引图像创建一幅RGB图像
mat2gray	通过数据缩放，而根据矩阵数据创建一幅灰度图像
rgb2gray	根据一幅RGB图像创建一幅灰度图像
rgb2ind	根据一幅RGB图像创建一幅索引图像

除了以上的标准转换方法之外，还可以利用某些函数返回的图像类型与输入的图像类型之间不同这一特点进行类型转换。例如，基于区域的操作函数总是返回二进制图像，用这些函数可以实现索引或灰度图像向二进制图像的转换。

MATLAB图像处理工具箱还提供了图像存储类型间的转换函数，这些函数包括im2double、im2uint8和im2uint16。这些函数可以自动进行原始数据的重新标度和偏移。这3个函数的调用格式非常简单，输入参数为图像矩阵，输出为转换后的图像。例如，以下命令将一个描述双精度RGB1图像的矩阵（数据范围为[0,1]）转换为一个uint8类型的RGB2图像矩阵（[0,255]范围内）。

```
RGB2=im2uint8(RGB1);
```

也可以使用MATLAB函数对图像存储类型进行转换。例如，double函数可以将uint8或uint16数据转换为双精度数据。存储类型间的转换将改变MATLAB及其工具箱理解图像数据的方式。如果用户希望转换后得到的数组能够被正确地理解为图像数据，那么在进行转换时需要重新标度或偏移数据。例如，如果将一个uint16类型的灰度图像转换为uint8类型的灰度图

像，那么函数 im2uint8 将对原始图形的灰度级进行量化，换句话说，所有 0～128 之间的原始图像数值都将变为 uint8 图像数据中的 0，而 129～385 之间的数值都为 1，以此类推。当使用一种位数较少的类型来描述数字图像时，通常有可能丢失用户图像的一些信息。例如，一个 uint16 类型的灰度图像能够存储 65 536 种不同的灰度级，但是一个 uint8 类型的灰度图像却只能存储 256 种不同的灰度级。一般这种信息的丢失不会产生严重的后果，因为 256 个灰度级仍然超过人眼所能分辨的色彩数目。

图像格式间的转换可以间接利用图像读写函数来完成：首先使用 imread 函数按照原有图像格式进行图像读取，然后调用 imwrite 函数对图像进行保存，并指定图像的保存格式。

例如，将一幅图像由 BMP 格式转换为 PNG 格式，则可以这样实现：首先使用 imread 读取 BMP 图像，然后调用 imwrite 函数来保存图像并指定为 JPG 格式：

```
bitmap=imred('my.bmp','bmp');
imwrite(bitmap,'my.jpg','jpg');
```

2.8 MATLAB 图像显示

读者已经在前面的一些例子中接触过 MATLAB 的图像读写和显示的操作。本节将进一步详细介绍如何使用 MATLAB 图像处理工具箱提供的图像读、写和显示函数。

2.8.1 MATLAB 图像的读写和显示

1. 读取图像

函数 imread 可以从任何 MATLAB 支持的图形图像文件格式中以任意位深度读取一幅图像，其基本调用格式如下：

```
[X, MAP]=imread(FILENAME, 'FMT')
```

其中，FILENAME 为需要读入的图像文件名。FMT 为图像格式（可以是 JPG/JPEG、TIF/TIFF、GIF、BMP、PNG、HDF、PCX、XWD、CUR 和 ICO），如果不指定 FMT 参数，那么系统将根据文件名自动判断图像类型。输出参数 X 表示存储图像数据的矩阵名，当图像为索引图像时，MAP 为该图像的调色板。在实际应用中，可以通过使用 imread 函数的在线帮助来获得最新的图像文件格式及其位深度信息。

imread 还可以分帧读取一个多帧图像文件。例如，以下语句将读取 mri.tif 文件的第 5 帧图像：

```
imread('mri.tif', 5);
```

大多数图像文件格式采用 8 位数据存储像素值，将这些文件读入内存后，MATLAB 都将其存储为 uint8 类型。对于支持 16 位数据的文件格式，例如，PNG 和 TIFF，MATLAB 将这些图像存储为 uint16 类型。和其他 MATLAB 生成的图像一样，一旦一幅图像被显示了，那么它将成为一个图形对象句柄。例如，以下代码将图像 pout.tif 读入 MATLAB 工作平台，读取数据矩阵为变量 My_I：

```
My_I=imread('pout.tif');
```

对于索引图像来说，即使调色板数据本身是 uint8 或 uint16 类型的，imread 函数也要将调色板读入一个双精度类型的数组中。

2. 写图像

函数 imwrite 可以将一幅图像写成一个 MATLAB 支持的格式图形文件。imwrite 函数基本的调用格式如下：

```
imwrite(X, MAP, FILENAME, 'FMT')
```

其中，X 为图像变量名，MAP 为调色板，FILENAME 为输出文件名，FMT 为指定的存储格式。如果用户指定文件名时包括了扩展名，那么 MATLAB 将根据这个扩展名推断所需的文件格式。imwrite 函数使用以下的规则来决定输出图像使用的存储种类：如果指定的输出图像文件格式支持 8 位图像，则创建一个 8 位图像文件；如果指定的输出图像文件格式支持 16 位图像，则创建一个位图像文件；如果指定的输出图像文件格式不支持 16 位图像，则用 uint8 来标度图像数据，并创建一个 8 位图像文件（这是因为大多数图像文件格式都使用 8 位）。

例如，以下语句将根据一个 MAP 文件（MATLAB 数据文件）装载一幅小丑图像，然后将其保存为一个包含小丑图像的 BMP 文件。

```
load clown
imwrite(X, map, 'clown.bmp');
```

对于某些图像格式，imwrite 函数还提供一些额外的参数来控制图像格式。例如，以下代码将一个灰度图像 I 写为一个位深度为 4 的 PNG 文件：

```
imwrite(I, 'clown.png', 'BitDepth', 4);
```

可以通过调用 imfinfo 函数获得与图像文件有关的信息，其调用格式如下：

```
INFO=imfinfo(FILENAME, 'FMT')
```

其中，INFO 是一个结构体，包含与文件的具体类型有关的文件信息，对于每一种类型的文件，INFO 都包含这样一些信息：文件名、文件格式、文件格式版本号、文件的修正数据、文件的字节大小、图像宽度（以像素为单位）、图像高度（以像素为单位）、每个像素所占的位数、图像类型等。

3. 显示图像

在 MATLAB 中，显示一幅图像的基本方法是使用 image 函数，这个函数将创建一个图形对象句柄，其调用格式如下：

```
image(X, Y, C)
```

其中，X 和 Y 表示图像显示位置的左上角坐标，C 表示需要显示的图像。另一个图像显示函数 imagesc 与 image 函数类似，但是它可以自动标度输入数据。

MATLAB 图像处理工具箱提供了一个高级的图像显示函数 imshow。和 image 与 imagesc 函数一样，这个函数也将创建一个句柄图形图像对象，所不同的是，imshow 函数将自动设置图形对象句柄的属性以及图像特征，从而获得最佳的显示效果。在图像处理应用程序中，使用 imshow 函数比使用 image 和 imagesc 函数要好，因为 imshow 函数的调用方法比较简单，并且在大多数情况下都是使用一个屏幕像素来显示一个图像像素的。

imshow 函数的基本调用格式如下：

```
imshow(C,MAP,Opt);
```

其中，C 表示待显示的图像，当 C 为索引图像时才需要 MAP 调色板参数；Opt 是可选参数，用来设置图像显示时是否显示为真实大小。

这里要注意的是，某些工具箱选项的设置会对 imshow 的显示效果产生影响，选项摘要包括下列 4 项。

ImshowBorder：图像显示时在图像坐标轴和窗口边界之间是否留有边框；

ImshowAxesVisible：是否显示图像的坐标轴及其标记；

ImshowTruesize：是否调用 truesize 函数。truesize 函数给为一个图像的像素单元分配一个单独的屏幕像素，也就是说，一个 100×200 的图像将显示为 100 个屏幕像素高，200 个屏幕像素宽；

TureSizeWarn：当图像大于屏幕大小时是否发出警告信息。

可以调用 iptsetpref 函数来设置工具箱的这些选项。例如，设置图形窗口大小恰好完全包含图像：

```
iptsetpref('ImshowBorder', 'tight');
```

在大多数情况下，imshow 函数在显示图像之前自动调用 truesize 命令；并将图像显示为真实大小，但是某些情况下，可能不希望调用 truesize 函数，而将图像显示为缺省的坐标轴尺寸。例如，当处理一幅小图像时会希望图像能够放大，在这种情况下，屏幕像素值与图像矩阵的元素之间并不是直接对应的，MATLAB 必须使用插值方法来决定屏幕像素的数值。

有两种方式可以决定 MATLAB 是否自动调用 truesize 命令。一种方法是调用中 iptsetpref 函数将 MATLAB 工具箱 ImshowTuresize 选项设置为 manual：

```
iptsetpref('ImshowTruesize', 'manual')
```

另一种方法就是通过设置 imshow 函数的 Opt 参数值，将其设置为 notruesize，这样在显示图像时将不会调用 truesize 命令.

如果还希望将本次运行中的设置保存到下次运行中保存到 startup.m 文件中。

2.8.2 二进制图像的显示方法

使用以下调用格式可以显示二进制图像：

```
imshow(BW)
```

在 MATLAB 中，一幅二进制图像是一个 uint8 或双精度类型的二维逻辑矩阵，该矩阵中元素的数值为 0 或 1。工具箱不支持 uint16 类型的二进制图像，所有能够返回二进制图像的工具箱函数都采用 uint8 逻辑数组存储图像数据。通常使用工具箱对二进制图像进行操作是非常简单明了的，在大多数情况下，用户装载一幅 1 位二进制图像，然后 MATLAB 将在内存中创建一个 unit8 逻辑图像。

数据全部都是 1 和 0 的图像并不总是代表数据描述的是一幅二进制图像。二进制图像的所有数据都必须是逻辑数据。因此，数据恰巧都是 0 和 1 的灰度图像将不会被理解为二进制图像。

在显示二进制图像时，尤其是对文字图像进行处理时，往往希望将图像进行逆转，用数值 0 表示白色，1 表示黑色。使用“~”操作符（取反）进行这项工作。

【例 2-8】对原始二进制文字图像 2-13 进行取反操作后，结果如图 2-14 所示。

```
I=imread('wrod213.bmp');
imshow(I);
figure,imshow(~I);
```

图 2-13 二进制文字图像



图 2-14 取反后的二进制文字图像

2.8.3　灰度图像显示方法

显示灰度图像最基本的调用格式如下：

```
imshow(I)
```

imshow 函数通过将灰度值标度为灰度级调色板的索引来显示图像。如果 I 是双精度型的，那么像素值将在[0.0,1.0]的范围内，0.0 显示为黑色，1.0 显示为白色。读者注意，在将一幅图像进行某种时或者通过某种计算得到一个图像矩阵时，得到数据类型一般为双精度。在直接显示图像之前必须确保它的值在[0.0,1.0]的范围内，才能正确显示。

灰度图像与索引图像在使用 m×3 大小的 RGB 调色板方面类似，正常情况下无需指定灰度图像的调色板。MATLAB 使用一个灰度级调色系统（R=G=B）来显示灰度图像。默认情况下，24 位颜色系统中调色板包含 256 个灰度级，在其他系统中包含 64 或 32 个灰度级。

Imshow 函数显示图像时，读者可以明确指定所使用的灰度级数目。例如，指定用 64 个灰度级显示图像 I：

```
Imshow(64)
```

MATLAB 自动对灰度图像进行标度以适合调色板的范围，因此对灰度图像可以使用自定义大小的调色板。在某些情况下，需要将超出数据范围的显示为一幅灰度图像。例如，对一幅图像进行滤波，输出数据的部分值将超出原始图像的数据范围，为了将这些超过惯例范围的数据显示为图像，读者可以直接指定数据范围，其格式如下：

```
imshow(I, [low high])
```

其中，low 和 high 参数分别为数据数组的最大值和最小值。如果使用空矩阵（[ ]）指定数据范围，则 imshow 将自动进行数据标度。

【例 2-9】对灰度图像 2-15 进行滤波，得到的图像数据超出范围，用 imshow 显示得到的图像，如图 2-16 所示。

```
I=imread('testpat1.tif');
imshow(I);
J=filter2([1 2; -1 -2], I);
figure,imshow(J,[]);
```



图 2-15　原始图像



图 2-16　滤波后超出数据范围的图像显示

2.8.4 索引图像的显示

使用 imshow 显示索引图像需要指定图像的矩阵和调色板，格式如下：

```
imshow(X, map)
```

对于 X 的每一个像素，imshow 显示存储在 map 相应中的颜色。图像矩阵中数值和调色板之间的关系依赖于图像矩阵的数据类型。如果图像矩阵中数值是双精度类型的，那么数值 1 将指向调色板的第一行，数值 2 指向第二行，依此类推。如果图像矩阵的数据类型是 uint8 或 uint16，那么有一个偏移量；数值 0 将指向调色板的第一行，数值 1 指向第二行，依此类推。

索引图像的每一个像素都直接映射为其调色板的一个入口。如果调色板的颜色数目多于图像的颜色数目，那么调色板中额外的颜色都将被简单地忽略掉；如果调色板包含的颜色数目小于图像数目，那么所有超出调色板颜色范围的图像像素都将被设置为调色板中的最后一个颜色。

2.8.5 RGB 图像的显示方法

RGB 图像又称为真彩色图像，它直接对颜色进行描述而不使用调色板，显示 RGB 图像的函数格式如下：

```
imshow(RGB)
```

RGB 是一个 $m \times n \times 3$ 的数组。对于 RGB 中的每一个像素(i,j)，imshow 显示数值(i,j,1:3)所描述的颜色。每个屏幕的像素使用 24 位颜色的系统就能直接显示真彩图像，因为系统分给每个像素的红、绿、蓝颜色分量分配 8 位。在颜色较少的系统中，MATLAB 将综合使用图像近似和抖动技术来显示图像。

2.8.6 多幅图像显示

很多时候，读者希望比较图像处理前后，或者不同处理方法的差异，往往需要将几幅图像显示在同一个窗口中，以便观察比较。达到这一目的有两种方法：一种方法是联合使用 imshow 函数和 subplot 函数；另一种方法是联合使用 subimage 和 subplot 函数。

subplot 函数将一个图形窗口划分为多个显示区域。Subplot 的调用格式如下：

```
subplot(m,n,p)
```

它将图形窗口划分为 $m \times n$ 个矩形显示区域，并将第 p 个显示区设为当前工作区域。

【例 2-10】将例 2-7 中的两幅图像显示在同一个图形窗口中，如图 2-17 所示。只需在例 2-7 的代码后添加如下代码即可。

```
figure,subplot(1,2,1),imshow(I);
subplot(1,2,2),imshow(I2);
```




图 2-17　同一窗口图形显示两幅图

第3章 图像的增强

图像增强是相对于图像识别、图像理解而言的一种前期处理。在图像信号的采集、输入等过程中，总会使得图像质量的下降。例如，按检测对象大小和形状的要求看，图像的边缘过于模糊；在相当满意的一幅图像上会发现多了一些不知来源的黑点或白点；图像的失真、变形等。总之，输入的图像在视觉效果和识别方便性等方面可能存在诸多问题，这类问题统称为“质量”问题。

图像增强的目的是采用一系列技术改善图像的效果或将图像转换成一种更适合于人或计算机进行分析处理的形式，主要是指按需要对图像进行适当的变换突出某些有用的信息，去除或削弱无用的信息，如改变图像对比度、去除噪声或强调边缘的处理等。本章主要介绍图像增强的一些基本方法——图像变换、直方图变换、灰度变换、图像平滑和图像锐化等。

3.1 图像变换增强

3.1.1 概述

计算机图像处理中，图像变换是一种为了达到某种目的（通常是从图像中获取某种重要信息）而对图像使用的一种数学技巧，经过变换后的图像将更为方便、容易地处理和操作。

从实际操作来看，图像变换就是对原图像函数寻找一个合适的变换核的数学问题。本质上来说，图像变换有着深刻的物理背景。例如，函数的一次傅立叶变换反映了函数在系统频谱上的频率分布。如果希望在频谱上作某些特定的处理，从而改变函数的某种特性（例如，图像增强），那么可以再对函数作二次傅立叶变换。

在图像变换中，应用最广泛的变换就是傅立叶变换，从某种意义上说，傅立叶变换就是函数的第二种描述语言，掌握了傅立叶变换，人们就可以在空域或频域中同时思考处理问题的方法，这是一种非常重要的能力，有时能够使用简单的方法来解决非常复杂的问题。除了傅立叶变换以外，还有一些其他很重要的图像变换方法，如DCT离散余弦变换、Radon变换和哈达码变换等。

3.1.2 傅立叶变换

1. 连续傅立叶变换

假设函数 $f(x)$为实变量 x 的连续函数，且在$(-\infty,+\infty)$内绝对可积，则 $f(x)$的傅立叶变换定义如下：

$$F(u)=\int_{-\infty}^{+\infty}f(x)e^{-j2\pi ux}dx \tag{3-1}$$

假设 $F(u)$可积，求$f(x)$的傅立叶反变换定义如下：

$$f(x)=\int_{-\infty}^{+\infty}F(u)e^{j2\pi ux}du \tag{3-2}$$

式（3-1）和（3-2）称为傅立叶变换对。傅立叶变换前的变量域 x 称为空间域，变换后的变量域 u 称为频域。在实际应用中，对这两个式子所作的建设条件几乎总是能成立的。

一个实函数的傅立叶变换通常是复函数，即 $F(u)$可以表示为：

$$F(u)=R(u)+jI(u) \tag{3-3}$$

式中

$$|F(u)|=\sqrt{R^2(u)+I^2(u)} \tag{3-4}$$

$$\theta(u)=\arctan[\frac{I(u)}{R(u)}] \tag{3-5}$$

幅度函数$|F(u)|$称为$f(x)$的傅立叶谱，$\theta(u)$为傅立叶变换的相角。振幅谱的平方一般称为$f(x)$的能量谱，即：

$$|F(u)|^2=R^2(u)+I^2(u) \tag{3-6}$$

傅立叶变换中出现的变量 u 称为频率变量，这个名称的来由是这样的：用欧拉公式可将指数 $e[-j2\pi ux]$表示成下面的形式：

$$e[-j2\pi ux]=e^{-j2\pi ux}=\cos(2\pi ux)-j\sin(2\pi ux) \tag{3-7}$$

如果将式（3-1）中的积分表示为离散项和的极限，则显然 $F(u)$是包含了正弦项和余弦项的无限项的和，而且 u 的每一个值确定了它所对应的正弦—余弦对的频率。

傅立叶变换很容易推广到二维的情况，假设函数 $f(x,y)$是连续可积的，且 $F(u,v)$可积，则存在如下的傅立叶变换对：

$$F(u,v)=\int_{-\infty}^{+\infty}f(x,y)e^{-j2\pi(ux+vy)}dxdy \tag{3-8}$$

$$f(x,y)=\int_{-\infty}^{+\infty}F(u,v)e^{j2\pi(ux+vy)}dudv \tag{3-9}$$

同样可以将二维函数的傅立叶变换写为如下的形式：

$$F(u,v)=R(u,v)+jI(u,v) \tag{3-10}$$

振幅谱为：

$$|F(u,v)|=\sqrt{R^2(u,v)+I^2(u,v)} \tag{3-11}$$

相角为：

$$\theta(u,v) = \arctan[\frac{I(u,v)}{R(u,v)}] \tag{3-12}$$

谱能量为：

$$|F(u,v)|^2 = R^2(u,v) + I^2(u,v) \tag{3-13}$$

2. 离散傅立叶变换

在计算机上使用的傅立叶变换通常都是离散形式的，即离散傅立叶变换（DFT）。使用离散傅立叶类型变换的根本原因有两个：一是 DFT 的输入、输出均为离散形式的，这使得计算机非常容易操作；二是因为计算 DFT 存在快速算法——快速傅立叶变换（FFT），因而计算比较方便。

假设对函数 $f(x)$ 在 N 个等间隔点处进行采样，得到离散化的函数 $f(m)$（$m=1,2,\cdots,N-1$），定义一维离散傅立叶变换对形式如下：

$$F(p) = \frac{1}{N}\sum_{m=0}^{N-1} f(m)e^{-j2\pi pm/N} \qquad p = 0,1,2\cdots,N-1 \tag{3-14}$$

$$f(m) = \sum_{p=0}^{N-1} F(p)e^{j2\pi pm/N} \qquad m = 0,1,2,\cdots,N-1 \tag{3-15}$$

数字图像是一幅二维的离散函数 $f(m,n)$。定义二维离散傅立叶变换 DFT 和反 DFT 的关系如下：

$$F(p,q) = \frac{1}{MN}\sum_{n=0}^{N-1}\sum_{m=0}^{M-1} f(m,n)e^{-j2\pi(pm/M+qn/N)} \qquad p = 0,1,\cdots,M-1$$

$$q = 0,1,\cdots,N-1 \tag{3-16}$$

$$f(m,n) = \sum_{p=0}^{M-1}\sum_{q=0}^{N-1} F(p,q)e^{j2\pi(pm/M+qn/N)} \qquad p = 0,1,\cdots,M-1$$

$$q = 0,1,\cdots,N-1 \tag{3-17}$$

3. 傅立叶变换的性质

二维傅立叶变换的有重要性质如表 3-1 所示（详细描述见有关信号处理方面的书籍）。傅立叶变换的这些性质在图像处理与识别中有着非常广泛的应用，但在使用时必须明确性质所表示的物理含义，否则容易出错。

【例 3-1】数字图像矩阵数据的显示及其傅立叶变换

构造一个函数矩阵，并使用二进制图像来显示，如图 3-1 所示。经傅立叶变换后的图像如图 3-2 所示。

```
f=zeros(30,30);
```

```
f(5:24,13:17)=1;
imshow(f, 'notruesize');
F=fft2(f,256,256); % 快速傅立叶变换算法只能处矩阵维数为 2 的幂次，f 矩阵不
                   % 是，通过对 f 矩阵进行零填充来调整
F2=fftshift(F);    % 一般在计算图 3-1 的图形函数的傅立叶变换时，坐标原点在
                   % 函数图形的中心位置处，而计算机在对图像执行傅立叶变换
                   % 时是以图像的左上角为坐标原点，所以使用函数 fftshift 进
                   %行修正，使变换后的直流分量位于图形的中心；
figure,imshow(log(abs(F2)),[-1 5],'notruesize');
```



图 3-1　矩阵 f 的二进制图像显示



图 3-2　二进制图像的傅立叶变换结果

表 3-1　二维傅立叶变换的性质

性　质	空　域	频　域
加法定理	$f(x,y)+g(x,y)$	$F(u,v)+G(u,v)$
位移定理	$f(x-a,y-b)$	$e^{-j2\pi(au+bv)}F(u,v)$
相似性定理	$f(ax,by)$	$\frac{1}{\lvert ab\rvert}F(\frac{u}{a},\frac{v}{b})$
卷积定理	$f(x,y)*g(x,y)$	$F(u,v)G(u,v)$
可分离乘积	$f(x)g(y)$	$F(u)G(v)$
旋转	$f(x\cos\theta+y\sin\theta,-x\sin\theta+y\cos\theta)$	$F(u\cos\theta+v\sin\theta,-u\sin\theta+v\cos\theta)$
Rayleigh 定理	$\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\lvert f(x,y)\rvert^2 dxdy=$	$\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}\lvert F(u,v)\rvert^2 dudv$

3.1.3 离散余弦变换

1. 离散余弦变换的定义

离散余弦变换（DCT）是利用傅立叶变换的对称性，采用图像边界的褶翻操作将图像变换为偶函数形式，然后对这样的图像进行二维离散傅立叶变换，变换后的结果将仅包含余弦项，故称之为离散余弦变换。DCT 可以将图像描述为不同幅值和频率的正弦值之和的形式。对于一幅典型的图像，DCT 有这样的性质：许多有关图像的重要可视信息都集中在 DCT 变换的一小部分系数中。因此，DCT 变化在图像压缩中非常有用，是有损图像压缩国际标准 JPEG 算法的核心。

一个 $M\times N$ 矩阵 A 的二维 DCT 定义如下：

$$C(p,q)=a(p)a(q)\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}A_{mn}\cos[\frac{\pi(2m+1)p}{2M}]\cos[\frac{\pi(2n+1)q}{2N}]\quad \begin{matrix}0\leqslant p\leqslant M-1\\0\leqslant q\leqslant N-1\end{matrix}$$

（3-18）

其中

$$a(p)=\begin{cases}\dfrac{1}{\sqrt{M}} & p=0\\ \sqrt{\dfrac{2}{M}} & 1\leqslant p\leqslant M-1\end{cases}\tag{3-19}$$

$$a(q)=\begin{cases}\dfrac{1}{\sqrt{N}} & q=0\\ \sqrt{\dfrac{2}{N}} & 1\leqslant q\leqslant N-1\end{cases}\tag{3-20}$$

数值 $C(p,q)$ 称为 A 的离散余弦变换系数。离散余弦变换是一个可逆变换，其逆变换的定义如下：

$$A_{mn}=\sum_{p=0}^{M-1}\sum_{q=0}^{N-1}a(p)a(q)C(p,q)\cos[\frac{\pi(2m+1)p}{2M}]\cos[\frac{\pi(2n+1)q}{2N}]\quad \begin{matrix}0\leqslant p\leqslant M-1\\0\leqslant q\leqslant N-1\end{matrix}$$

（3-21）

DCT 逆变换可以理解为：任意一个 $M\times N$ 矩阵 A 都可以写成 $M\times N$ 个如式（3-22）所示的函数之和形式：

$$a(p)a(q)C(u,v)\cos[\frac{\pi(2m+1)p}{2M}]\cos[\frac{\pi(2n+1)q}{2N}]\quad \begin{matrix}0\leqslant p\leqslant M-1\\0\leqslant q\leqslant N-1\end{matrix}\tag{3-22}$$

这些函数被称为 DCT 基本函数。DCT 系数 $C(p,q)$ 可以看成是应用于每个函数的权值。

2. DCT 和 JPEG

DCT 是 JPEG 压缩算法的基础，以下将对 DCT 在 JPEG 中的应用做一个简单的介绍，详细内容可参见相关书籍。在 JPEG 图像压缩算法中，输入图像被分为 8×8 或 16×16 块，然后

对每一个块计算二维 DCT，接着再对 DCT 系数进行量化、编码和发送。JPEG 接收者通过对 DCT 系数进行解码，计算每个块的二维逆 DCT，然后将所有的块重新组装成一幅图像。对于一般的图像来说，DCT 系数许多都是接近于 0 的数值，可以丢弃这些系数而不会对图像的重建质量产生重大影响。

图像处理工具箱提供两种不同的方法计算 DCT：第一种方法是使用函数 dct2，该函数使用一个基于 FFT 的算法来提高当输入较大的输入方阵时的计算速度。dct2 函数的调用格式如下：

```
B=dct2(A, [M N])
```

或

```
B=dct2(A, M, N)
```

其中，A 表示要变换的图像，M 和 N 是可选参数，表示填充后的图像矩阵大小。B 表示变换后得到的图像矩阵。

第二种方法使用由函数 dctmtx 返回的 DCT 变换矩阵，这种方法较适合于较小的输入方阵（例如，8×8 或 16×16 方阵）。dctmtx 的调用格式如下：

```
D=dctmtx(N)
```

其中，N 表示要变换的矩阵，D 为变换后得到的图像矩阵。

【例 3-2】将一幅图像分成 8×8 个块进行二维离散余弦变换，然后丢弃块中近似于 0 的数值，最后对每一个块使用二维逆离散余弦变换重新构造图像。压缩前后的图像分别如图 3-3 和图 3-4 所示。



图 3-3　压缩前的原始图像



图 3-4　DCT 变换压缩后的图像

实现代码如下：

```
I=imread('cameraman.tif');                % MATLAB 自带的图像，如图 3-3 所示
imshow(I);
clear;close all
I=imread('cameraman.tif');
imshow(I);
I=im2double(I);
T=dctmtx(8(;
B=blkproc(I,[8 8], 'P1*x*P2',T,T');
Mask=[1 1 1 1 0 0 0 0
      1 1 1 0 0 0 0 0
```

```
        1 1 0 0 0 0 0 0
        1 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0
        0 0 0 0 0 0 0 0];
B2=blkproc(B,[8 8],'P1.*x',Mask);      % 此处为点乘(.*)
I2=blkproc(B2,[8 8],'P1*x*P2',T',T);
figure,imshow(I2);                    % 重建后的图像如图 3-4 所示
```

3.2 灰度变换增强

一般成像系统只具有一定的亮度范围，亮度的最大值与最小值之比称为对比度。由于形成图像的系统亮度有限，常出现对比度不足的问题，使人眼观看图像时视觉效果很差。通过灰度变换可以改善视觉效果。灰度变换又可以分为以下 3 种：线性变换、分段线性变换和非线性变换。

3.2.1 线性灰度变换

在曝光不足或过度的情况下，图像灰度可能会局限在很小的范围内。这时人们看到的将是一个模糊不清、似乎没有灰度层次的图像。用一线性单值函数，对图像内的每一个像素作线性扩展，将有效地改善图像视觉效果。

令原图像 $f(m,n)$的灰度范围为$[a,b]$，线性变换后，图像 $g(m,n)$的范围为$[c,d]$，如图 3-5 所示。$g(m,n)$和 $f(m,n)$之间的变换关系为：

$$g(m,n)=\frac{d-c}{b-a}(f-a)+c \tag{3-23}$$

从灰度直方图分析，由于$|d-c|$大于$|b-a|$，所以对离散图像来说，尽管变换前后像素的个数不变，但是不同像素之间的灰度差变大，对比度加大，图像质量优于变换前。对于连续图像，如果背景与目标物的灰度之差很小，在$[a,b]$区间内量化时可能进入同一灰度级内而不能分辨。如果线性变换时使

$$|d-c|=G|b-a|,\quad G\leqslant 2 \tag{3-24}$$

则量化时，$f(m,n)$变换后在$[c,d]$区间内就可以取 $G\times N$ 个以上不同的灰度值，N 是 $f(m,n)$在$[a,b]$区间内所取的灰度级个数，人的眼睛原本不能检测的目标用增强手段可以显示出来。

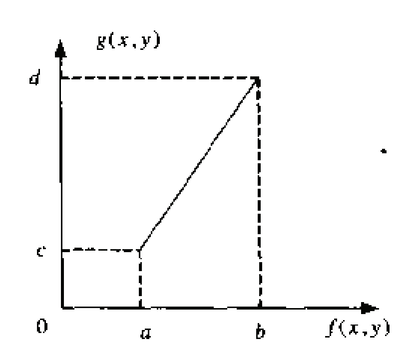


图 3-5　灰度线性变换示意图

若图像中大部分像素的灰度级在[a,b]范围内，少部分像素分布在小于 a 和大于 b 的区间内。此时可用式（3-25）作变换，如图 3-6 所示。

$$g(m,n)=\begin{cases} c & f(m,n)<a \\ c+\dfrac{c-d}{b-a}(f-a) & a\leqslant f(m,n)<b \\ d & f(m,n)\geqslant b \end{cases} \tag{3-25}$$

但是，这种两端“截取式”的变换使小于灰度级 a 和大于等于灰度级 b 的像素强行压缩为 c 和 d，将会造成一小部分信息丢失。但在实际应用中，如遥感资料分析降水时，在预处理中去掉非气象的信息，即可减少运算量又可提高分析精度。

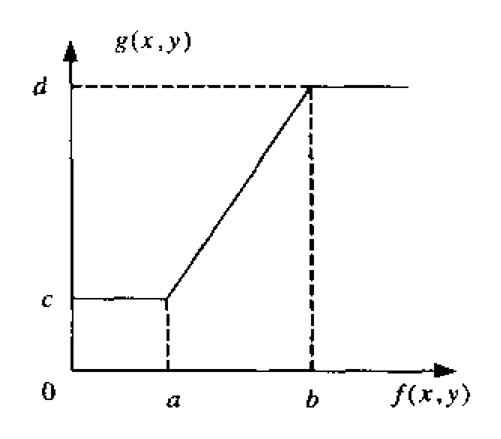


图 3-6　截取式线性变换示意图

3.2.2　分段线性变换

将图像灰度区间分成两端乃至多段分别作线性变换。分段线性变换的优点是可以根据用户的需要，拉伸特征物体的灰度细节，相对抑制不感兴趣的灰度级。如图 3-7 所示是分成 3 段的示意图，其数学表达式如下。

$$
g(m,n)=\begin{cases}\dfrac{c}{a}f(m,n) & 0\leqslant f(m,n)<a \\ \dfrac{d-c}{b-a}[f(m,n)-a]+c & a\leqslant f(m,n)<b \\ \dfrac{M_g-d}{M_f-b}[f(m,n)-b]+d & b\leqslant f(m,n)\leqslant M_f\end{cases} \tag{3-26}
$$

图中对区间$[a,b]$进行了灰度扩展，而区间$[0,a]$和$[b,M_f]$受到了压缩。通过调整折线拐点的位置及控制分段直线的斜率，可对任一灰度区间进行扩展或压缩。这种变换适用于在黑色或白色附近有噪声干扰的情况。例如照片中的划痕，由于变换后$[0,a]$和$[b,M_f]$受到了压缩，因而使污斑减弱。

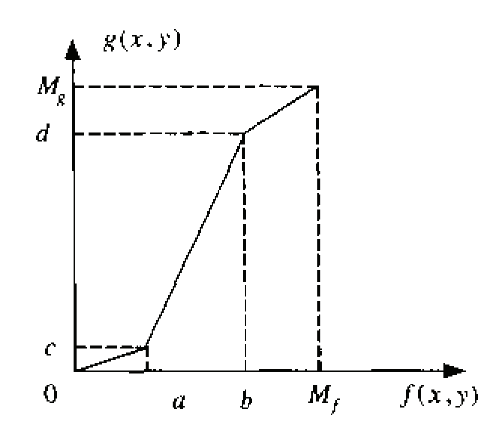


图 3-7　分段线性变换示意图

3.2.3 非线性灰度变换

当用某些非线性函数，例如对数函数、指数函数等作为映射函数时，可实现图像灰度的非线性变换。对数变换一般为：

$$
g(m,n)=a+\frac{\ln[f(m,n)+1]}{b\cdot\ln c} \tag{3-27}
$$

其中 a、b 和 c 是可调参数，用于调整曲线的位置和形状而引入的参数，它使图像的低灰度区得以扩展，而高灰度区得到压缩，以使图像的灰度分布均匀，与人的视觉特性匹配。

指数变换的一般式为：

$$g(m,n) = b^{c[f(m,n)-a]} - 1 \tag{3-28}$$

其中 a、b 和 c 参数用来调整曲线的位置和形状，它的效果与对数相反，它将对图像的高灰度区进行扩展。

MATLAB 的图像处理可以使用 imadjust 函数来实现图像的灰度变换，调用格式如下：

```
J=imadjust(I, [low_in high_in], [low_out high_out])
```

其中 low_in 和 high_in 指定输入图像需要调整的灰度范围，low_out 和 high_out 指定输出图像的灰度范围。

【例 3-3】采用灰度变换的方法增强图像的对比度。变换前的原始图像和灰度直方图如图 3-8 和图 3-9 所示，变换后的图像及相应灰度直方图如图 3-10 和图 3-11 所示。

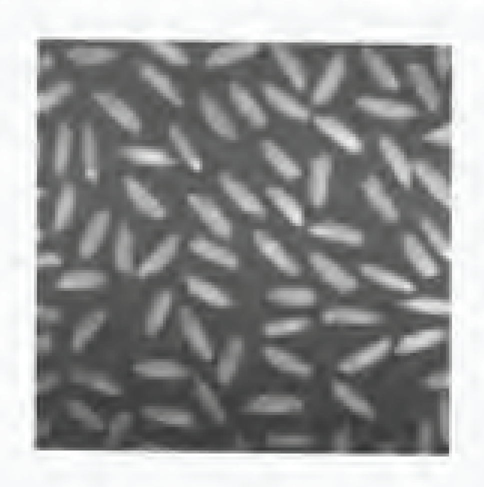


图 3-8　rice.tif 的原始图像

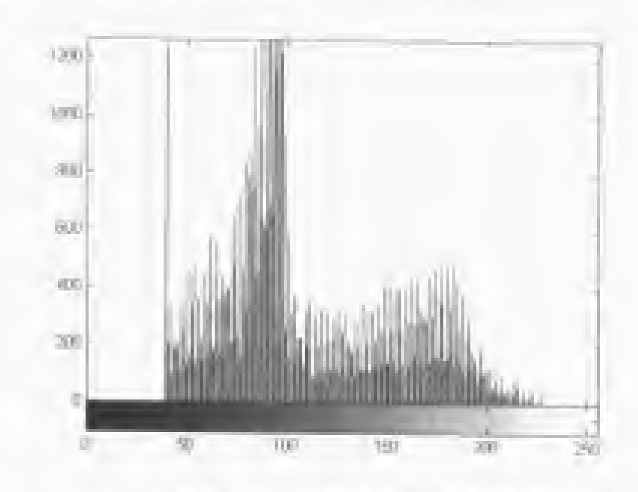


图 3-9　rice.tif 原始图像的灰度直方图



图 3-10　灰度变换后的图像

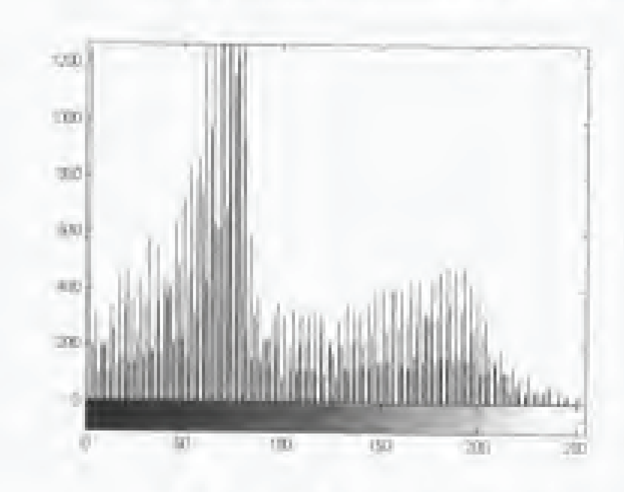


图 3-11　灰度变换后的直方图

实现代码如下：

```
I=imread('rice.tif');
imshow(I);
figure,imhist(I);
J=imadjust(I,[0.15 0.9], [0 1]);
```

```
figure,imshow(d);
figure,imhist(d);
```

通过灰度变换，使原来灰度范围集中在[40,225]之间的图像，变为灰度分布在[0,255]的整个区间，增强了图像的对比度，视觉效果得到改善。

从以上的例子可以看出，使用 imadjust 函数的灰度变换必须按照以下两个步骤来进行：

（1）绘制直方图，观察灰度范围；

（2）将灰度范围转换为 0.0～1.0 之间的分数，使得灰度范围[low_in,high_in]传递给 imadjust 函数。

在 MATLAB 中，还可以使用 stretchline 函数，它能够计算图像的直方图，并自动判断调节范围。Stretchline 函数以分向量形式返回灰度范围，然后将这个向量直接传递给 imadjust 函数。

3.3 直方图变换增强

3.3.1 灰度直方图

面积为 A 的连续图像 $f(x,y)$经数字化后，成为 M 行 N 列的数字图像 $f(m,n)$。一般而言，在数字图像 $f(m,n)$中取不同灰度值的像素的数目是不同的。直方图是用于表达图像灰度分布情况的统计图表。对数字图像 $f(m,n)$而言，其横坐标是灰度值 r，纵坐标是出现这个灰度值的概率值 $p(r_i)$。

设数字图像像素的灰度值为 $r_0,r_1,\cdots,r_{L-1}$，则概率 $p(r_i)$为：

$$p(r_i)=\frac{n(r_i)}{N} \qquad i=0,1,\cdots,L-1 \tag{3-29}$$

其中 $n(r_i)$是灰度为 r_i 的像素点总数，N 是一幅图像的总像素点，且

$$\sum_{i=0}^{L-1} p(r_i)=1 \tag{3-30}$$

尽管灰度直方图不能表示出有某灰度级的像素在什么位置，更不能直接显示图像内容，但是具有统计特性的直方图却能描述该图像的灰度分布特性，使人们从中得到的诸如总体明亮程度、对比度、对象物的可分性等与图像质量有关的灰度分布概貌，成为一些处理方法的重要依据。

在 MATLAB 中，imhist 函数可以用来计算和绘制图像的直方图，该函数的调用形式为：

imhist(I,N)——在长度为 N 的灰度条上显示灰度图像 I 的直方图，对于灰度图像 N 的缺省值为 256，对于黑白二值图像，N 的缺省值为 2。

mhist(X,MAP)——在色彩板 MAP 的色彩条上显示附标图像 X 的直方图，色彩板必须至少和 X 中的附标具有一样的长度。

[counts,X]=imhist(···)——对于灰度图像返回每个灰度上的像素点的数目，对附标图像则返

回色彩板对应的像素点数。

有关 imhist 函数的应用举例，读者可以参考前面的灰度变换或者后续的直方图变换，此处不再赘述。

3.3.2　直方图均匀化

直方图均匀化是另一种灰度增强的算法。一副对比度小的图像，其直方图分布一定集中在某一比较小的范围内，经过均匀化处理的图像，其所有灰度级出现的概率相同，此时图像的熵最大，图像所包含的信息量最大。图 3-12 所示为连续情况下非均匀概率密度函数 $P_r(r)$经过函数 $T(r)$转换为均匀概率密度分布 $P_s(s)$的情况。其中，r 为变换前的规一化灰度级，$0 \leqslant r \leqslant 1$。$s=T(r)$为变换后的灰度值，也归一化为 $0 \leqslant s \leqslant 1$。若（1）在 $0 \leqslant r \leqslant 1$ 区间内，$T(r)$为单调递增函数，且满足 $0 \leqslant T(r) \leqslant 1$。（2）反变换 $r=T^{-1}(s)$存在，$0 \leqslant s \leqslant 1$，也满足类似（1）的条件，变换后的图像灰度范围与原图一致。

变换前后满足：

$$P_s(s)ds = P_r(r)dr \tag{3-31}$$

则：

$$P_s(s) = [P_r(r)\frac{dr}{ds}]_{r=T^{-1}(s)} \tag{3-32}$$

直方图归一化后有 $P_s(s)=1$，而 $s=T(r)$，则代入式（3-32）：

$$\frac{ds}{dr} = P_r(r) = \frac{dT(r)}{dr} \tag{3-33}$$

两边取积分得：

$$s = T(r) = \int_0^r P_r(r)dr \tag{3-34}$$

式（3-34）为所求的变换函数，它是原始图像的累计分部函数（CDF），是一个非负的递增函数。

在离散情况下，像素点的总数为 N，L 个灰度等级，第 i 个灰度级 r_i 出现的频数为 $n(r_i)$，$0 \leqslant r_i \leqslant 1$，$i=0,1,\cdots,L\text{-}1$。则对其进行均匀化的变换函数为：

$$T(r_i) = \sum_{j=0}^{i} \frac{n(r_j)}{N} \tag{3-35}$$

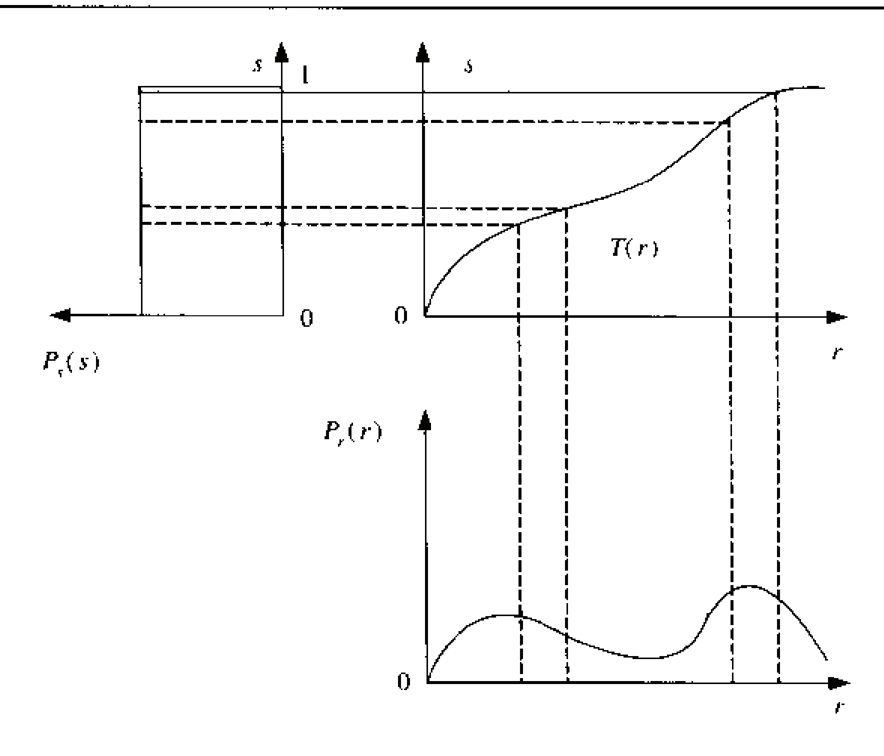


图 3-12 将非均匀概率密度变换为均匀概率密度

从理论上说，直方图均匀化是通过变换函数将原始图像的直方图调整为平坦的直方图，然后用此均匀直方图校正图像。但实际上直方图均匀化后的图像直方图并不是完全均匀的，因为操作过程中原直方图上频数小的某些灰度级要求并入一个或几个灰度级中。

3.3.3 直方图均匀化的计算步骤及实例

直方图均匀化的计算步骤如下：

（1）列出原始图像的灰度级 r_i，$i=0,1,\cdots,L-1$，其中 L 是灰度级的个数；

（2）统计各灰度级的像素数目 $n(r_i)$，$i=0,1,\cdots,L-1$。

（3）计算原始图像直方图各灰度级的频数 $P(r_i)=n(r_i)/N$，$i=0,1,\cdots,L-1$，其中 N 为原始图像总的像素数目。

（4）计算累计分布函数 $T(r_i)=\sum_{j=0}^{i}P(r_j)$，$i=0,1,\cdots,L-1$。

（5）应用以下公式计算映射后的输出图像的灰度级 g_i，$i=0,1,\cdots,P-1$，P 为输出图像灰度级的个数：

$$g_i = INT\left[(g_{\max} - g_{\min})T(r_i) + g_{\min} + 0.5\right] \tag{3-36}$$

其中，INT 为取整符号。

（6）统计映射后灰度级的像素点数量 $n(g_i)$，$i=0,1,\cdots,P-1$。

（7）计算输出图像直方图 $P(g_i)=n(g_i)/N$，$i=0,1,\cdots,P-1$。

（8）用 r_i 和 g_i 的映射关系调整原始图像的灰度级，获得直方图均匀分布的输出图像。

MATLAB 中，可以使用 histeq 函数实现直方图均匀化。函数调用方法为：

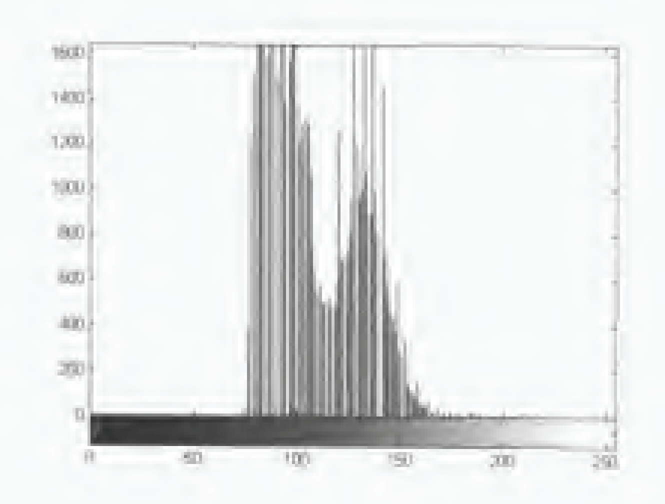
```
[J,T]=histeq(I,N)
```

该函数对图像 I 进行变换，返回有 N 级灰度的图像 J，J 中的每个灰度级具有大致相等的像素点，所以图像 J 的直方图比较平坦，N 的默认值为 64。T 是转移函数。

【例 3-4】对 3-13(a)所示的 pout.tif 图像进行直方图均匀化。图 3-13(b)为图 3-13(a)的灰度直方图，经直方图均匀化后的 64 级灰度图像及相应灰度直方图如图 3-14(a)和 3-14(b)所示，相应灰度变换曲线如图 3-14(c)所示。均匀化后的 32 级灰度图像如图 3-15(a)和 3-15(b)所示。



(a) pout.tif 原始图像



(b) 原始图像的灰度直方图

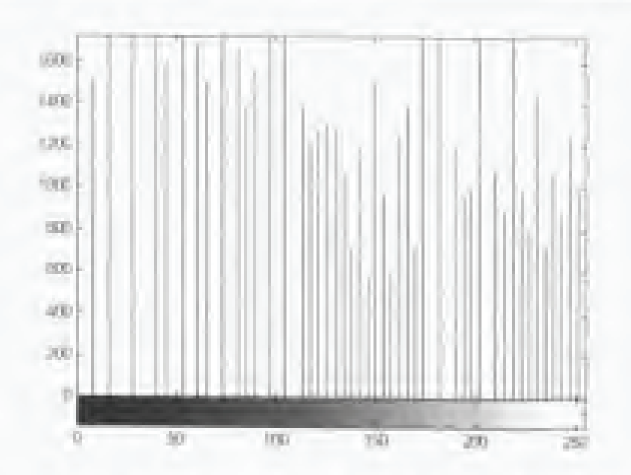
图 3-13

实现代码如下：

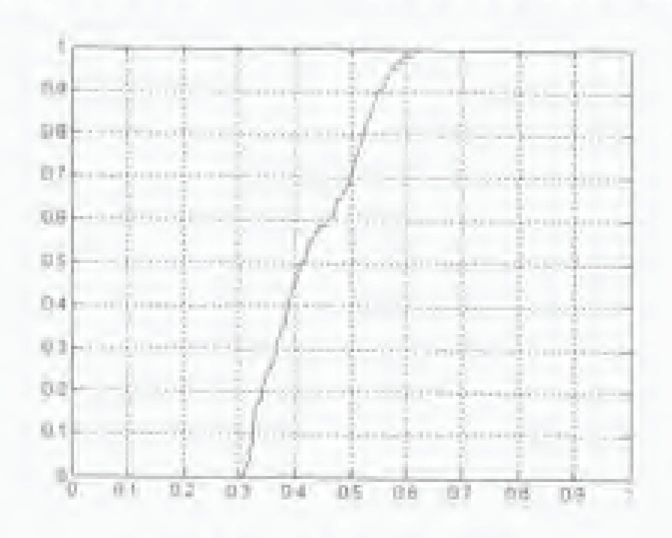
```
I=imread('pout.tif');  % 读取 MATLAB 自带的 pout.tif 图像
imshow(I);
figure,imhist(I);
[J,T]=histeq(I,64);      % 图像灰度扩展到 0~255，但是只有 64 个灰度级
figure,imshow(J);
figure,imhist(J);
figure,plot((0:255)/255,T);  % 转移函数的变换曲线
J=histeq(I,32);
figure,imshow(J);   % 图像灰度扩展到 0~255，但是只有 32 个灰度级
figure,imhist(J);
```



(a) 直方图均匀化后的 64 级灰度图像



(b) 直方图均匀化后的 64 级灰度图像的直方图

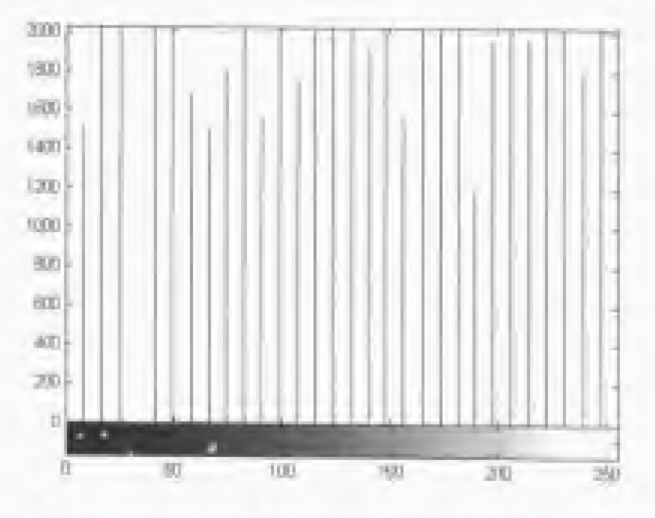


(c) 直方图均匀化的变换曲线 $T(r)$

图 3-14



(a) 直方图均匀化后的 32 级灰度图像



(b) 直方图均匀化后的 32 级灰度图像的直方图

图 3-15

3.4　空间域滤波增强

图像实质是光电信息，因此图像噪声的主要来源有以下 3 个：在光电、电磁转换过程中引入的人为噪声；大气层电（磁）暴、闪电、电压和浪涌等引起的强脉冲行冲击干扰；自然起伏性噪声，由物理的不连续性或粒子性所引起。噪声恶化了图像质量，使图像模糊，甚至淹没和改变特征，给图像分析和识别带来的困难。

为了消除噪声，常采用滤波的方法，它可以分为空域滤波和频域滤波，本节主要介绍空域滤波。按照空域滤波器的功能又可以分为平滑滤波器和锐化滤波器。平滑滤波器可以用低通滤波实现，目的在于模糊图像（提取图像中的较大对象而消除小对象或将对象的小间断连接起来）或消除图像噪声；锐化滤波器是用高通滤波实现的，目的在于强调图像被模糊的细节。下面分别介绍平滑滤波器和锐化滤波器的使用方法和用途。

3.4.1　平滑滤波器

1. 邻域平均法

邻域平均法是一种局部空间域处理的算法。设一幅数字图像 $f(x,y)$ 为 $M\times N$ 的阵列，平滑后的图像为 $g(x,y)$，它的每个像素的灰度由包含 (x,y) 的预定邻域的几个像素的灰度级的平均值所决定，即用下式得到平滑图像。

$$g(x,y)=\frac{1}{K}\sum_{(i,j)\in S}f(i,j) \tag{3-37}$$

式（3-37）中 $x=1,2,\cdots,M$，$y=1,2,\cdots,N$，S 是 (x,y) 像素点的预定邻域（不包括 (x,y) 像素点），K 是 S 内的坐标点总数。

设噪声 $e(x,y)$ 是加性白噪声，均值为 0，方差为 σ^2，而且噪声与图像 $f(x,y)$ 互不相关。噪声干扰的图像为：

$$g(x,y)=f(x,y)+e(x,y) \tag{3-38}$$

经邻域平均法处理后的图像 $g(x,y)$ 为：

$$g(x,y)=\frac{1}{K}\sum_{(i,j)}\sum_{\in S}f(i,j)+\frac{1}{K}\sum_{(i,j)}\sum_{\in S}e(i,j) \tag{3-39}$$

处理后残余噪声的平均值为：

$$E[\frac{1}{K}\sum_{(i,j)}\sum_{\in S}e(i,j)]=0 \tag{3-40}$$

残余噪声的方差为：

$$D[\frac{1}{K}\sum_{(i,j)}\sum_{\in S}e(i,j)]=\frac{1}{K}\sigma^2=0 \tag{3-41}$$

上式表明邻域平均处理后，残余噪声的方差减小为原来的 $1/K$。但是图像由 $f(x,y)$ 变为

$\frac{1}{K}\sum_{(i,j)}\sum_{\in S} f(i,j)$，这将会引起图像中目标对象的轮廓变模糊或细节特征消失。

最典型的邻域 S 有两种，分别为 4 邻域和 8 邻域，如图 3-16(a)和图 3-16(b)所示。用模板表示邻域平均算法，4 邻域和 8 邻域的模板表示如下：

$$M_4 = \frac{1}{4}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{3-42}$$

$$M_8 = \frac{1}{8}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3-43}$$

模板沿水平和垂直两个方向逐点移动，从而平滑整幅图像。模板内各系数之和为 1，用这样的模板处理常数图像时，图像没有变化；对一般图像处理后，整幅图像灰度的平均值可以保持不变。



(a) 像素点点的 8 邻域　　　(b) 像素点点的 4 邻域

图 3-16

2. 选择平均法

选择平均法是以领域平均法为基础的。它只对灰度值相同或相近的像素进行平均，或者按照灰度特殊的程度加权之后再求和，以免造成目标边缘的模糊。

例如 3×3 邻域内的图像为：

$$\begin{matrix} A_1 & A_2 & A_3 \\ B_1 & B & B_2 \\ B_3 & B_4 & B_5 \end{matrix}$$

其中 A_1、A_2和 A_3的值相近，而 B 与 $B_1,B_2,\cdots,B_5$的值相近，这表明 A_i与 B_j之间有边缘存在，因而用模板：

$$\frac{1}{6}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

进行处理，使得与 B 相差比较大的 A_i的值不会混入平均值中去，得到不模糊边缘的效果。这就是选择平均法。模板中的系数不是 0 就是相等的常数。为了使处理后的图像的平均灰度值

不变，模板中各系数之和应为 1。

模板的大小一般取 3×3。如果模板大，去噪声效果更明显，但是计算复杂，所涉及的像素多而容易把细节抹去，造成模糊。在 3×3 模板去噪声效果不明显时，可以重复迭代几次，并监视中间结果。经常用的选择平均法有阈值法和半邻域法。

（1）阈值法

阈值法的数学表达式如下：

$$g(x,y)=\begin{cases}\dfrac{1}{M}\sum\limits_{(i,j)\in S}f(i,j) & \left|f(x,y)-\dfrac{1}{M}\sum\limits_{(i,j)\in S}f(i,j)\right|>T\\ f(x,y) & \text{其他}\end{cases} \tag{3-44}$$

其中 S 是点(i,j)的一个邻域，M 是所取邻域内的像素点数量，T 是一个预先设定的非负阈值。当一些像素和它的邻域值的差值小于规定的阈值 T 时，仍保留这些像素的灰度值。这样平滑后的图像比邻域平均值法模糊度降低。当某些像素的灰度值与其邻域像素的灰度值均值差别较大时，它必然是噪声，则取邻域平均值作为该像素的灰度值，仍具有较好的平滑效果。

（2）半邻域法

设在 3×3 窗口的局部区域内，图像半邻域法的具体算法为：

（1）对 A_i 排序，灰度值较大的前 5 点构成 B 组，灰度值较小的后 3 点构成 A 组；

（2）设定门限值 T；

（3）求 A、B 两组的平均值 $\overline{A}$ 和 $\overline{B}$；

（4）若$\left|\overline{A}-\overline{B}\leqslant T\right|$认为无边缘通过，用 8 邻域平均；反之，则认为有边缘通过，P 与 B 组中的 5 点进行 6 点平均。

3. 中值滤波

中值滤波是一种非线性滤波技术，由于实际计算过程中并不需要图像的统计特性，所以比较方便。它是基于图像的这样一种特性：噪声往往以孤立的点的形式出现，这些点对应的像素数很少，而图像则是由像素数较多、面积较大的小块构成。在一定条件下，可以克服线性滤波器所带来的图像细节模糊，而且对滤除脉冲干扰及图像的扫描噪声最为有效。但是对一些细节多，特别是点、线和尖顶多的图像不宜采用中值滤波的方法。

在一维的情况下，中值滤波器是一个含有奇数个像素的窗口。在处理之后，将窗口正中的像素灰度值，用窗口内各像素灰度值的中值代替。例如若窗口长度为 5，窗口中像素的灰度值为 80、90、200、110 和 120，则中值为 110。因为按小到大（或大到小）排序后，第三位的值是 110，于是原来窗口正中的灰度值 200 就由 110 取代。如果 200 是一个噪声的尖峰，则将被滤除。然而，如果它是一个信号，则滤波后就被消除，降低了分辨率。因此中值滤波在某些情况下抑制噪声，而在另一些情况下却会抑制信号。

设有一个一维序列 $f_1,f_2,\cdots,f_n$。取窗口长度为奇数 m（m 为奇数），对此序列进行中值滤波，就是从输入序列中相续抽出 m 个数，$f_{i-v},\cdots,f_i,\cdots,f_{i+v}$，其中 f_i 为窗口的中心值，$v=(m-1)/2$，再将这 m 个点的数值按其数值大小排列，取其序号为正中间的那个数作为滤波输出。中值滤波表达式为：

$$Y_i = Med\{f_{i-v} \cdots f_i \cdots f_{i+v}\} \quad i \in Z \quad v = \frac{m-1}{2} \tag{3-45}$$

对数字图像进行中值滤波，实质是对二维序列$\{X_{ij}\}$的中值滤波，滤波窗口也是二维的，但这种二维窗口可以有各种不同的形状，如线状，方形、圆形、十字形和原环形等。二维的中值滤波可以表示为：

$$Y_{ij} = \sum_{A} Med\{X_{ij}\} \quad A\text{为窗口} \tag{3-46}$$

在实际使用中，窗口的尺寸一般先选 3 再选 5，逐渐增大，直到滤波效果满意为止。对于有缓慢变化的较长轮廓线物体的图像，采用方形或圆形窗口为宜，对于包含尖顶角的图像，适宜用十字形窗口。但是，一定要注意保持图像中有效的细状物体。

4. 自适应滤波

考虑到图像 $f(x,y)$中目标对象和背景一般具有不同的统计特性，即有不同的均值和方差。因此，为了尽可能保留关键的边缘信息，可采用动态的或自适应的局部平滑滤波，这样如果目标对象比较大时，能够得到较满意的图像细节特性。

图像的噪声消除可以采用自适应魏纳滤波。魏纳滤波器将图像信号和噪声都看成随机信号，在对随机信号进行分系统计的基础上设计出符合最优准则的滤波器。假设图像 $g(x,y)$是由真实图像 $f(x,y)$和噪声 $n(x,y)$构成的，设计图像滤波器的目的就是使输出的图像 $I(x,y)$尽可能地降低噪声信号 $n(x,y)$，同时恢复真实图像$f(x,y)$。定义误差信号：

$$e(x,y) = f(x,y) - I(x,y) \tag{3-47}$$

均方误差是平均误差的度量：

$$MSE = \sum_{x=1}^{M}\sum_{y=1}^{N} e^2(x,y) \tag{3-48}$$

魏纳滤波器以最小化均方误差作为最优准则，设计滤波器。有关魏纳滤波器设计的原理和设计方法的理论推倒比较复杂，请读者参考有关文献。

3.4.2 空间域图像平滑实例

由于图像中噪声空间相关性比较弱，而且噪声频谱一般位于空间频率较高的区域，而图像本身的频率分量则处于较低的空间频率域内，因此可以用低通滤波的方法来实现平滑。

图像增强操作主要是针对图像的各种噪声而言的，为了说明以上滤波方法的用途，需要模拟数字图像的各种噪声来分析滤波效果。数字图像产生噪声的途径有很多种，具体依赖于图像的数字化方式。例如，如果图像是由电影图片扫描而成的，那么电影颗粒就是一个噪声源。噪声也可以是电影遭到破坏的结果，或者是由扫描仪自身产生的。如果图像直接以数字形式获得，那么收集数据的机制（例如，CCD 扫描仪）将会不可避免地引入噪声。另外，图像数据的传输也会引入噪声。

MATLAB 的图像处理工具箱提供 imnoise 函数，可以用该函数给图像添加不同种类的噪声，该函数的调用格式如下：

```
J=imnoise(I,'tyep',parameters);
```

其中I为加噪声前的图像，J为加噪声后的图像，type为噪声类型。表3-2列出了imnoise函数能够产生的5种噪声及其对应参数。

表3-2　　imnoise函数支持的噪声类型及参数说明

类　型	参　数	说　明
gaussian	m,v	均值为m，方差为v的高斯噪声
localvar	v	均值为0，方差为v的高斯噪声
possion	无	泊松噪声
Salt & pepper	d	密度为d的椒盐噪声
speckle	v	均值为0，方差为v的均匀分布的随机噪声

【例3-5】对原始图像3-17(a)分别叠加高斯噪声和椒盐噪声，叠加噪声后的图像如图3-17(b)和3-17(c)所示。

```
I=imread('eight.tif');
imshow(I) ;

J1=imnoise(I,'gaussian',0,0.02); % 叠加均值为0，方差为0.02的高斯噪声，可以用
                                 % localvar代替，如图3-17(b)所示
figure,imshow  (J1);

J2=imnoise(I,'salt & pepper',0.04); % 叠加密度为0.04的椒盐噪声。
                                    % 如图3-17(c)所示。
figure,imshow(J2);
```



(a) 未加噪声的原始图像



(b) 高斯噪声干扰的图像



(c) 椒盐噪声干扰的图像

图3-17

【例3-6】采用MATLAB中的二维中值滤波函数为medfilt2对受椒盐噪声干扰的图3-17(c)采用二维中值滤波虑除噪声，窗口的大小分别选择为3、5和7。中值滤波后的图像如图分别如图3-18(a)、3-18(b)和3-18(c)所示。

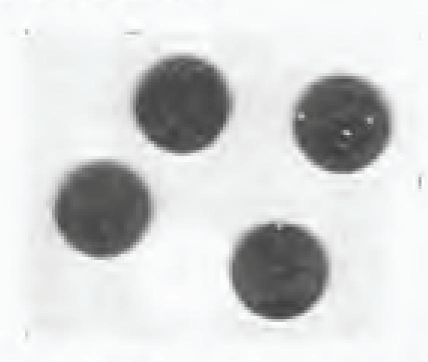
```
I_Filter1=medfilt2(J2,[3 3]);  %窗口大小为3×3，结果如图3-18(a)所示
figure,imshow(I_Filter1);
I_Filter2=medfilt2(J2,[5 5]);  %窗口大小为5×5，结果如图3-18(b)所示
```

```
figure,imshow(I_Filter2);
I_Filter3=medfilt2(J2,[7 7]);  %窗口大小为7×7，结果如图3-18(c)所示
figure,imshow(I_Filter3);
```

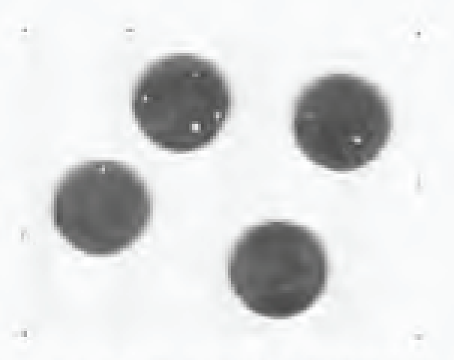
观察、比较滤波后的图像比较不同窗口滤波后的图像，可以发现窗口越大，图像中的细节丢失越多，尤其是物体的边缘会变得更加模糊。



(a)大小为 3×3 的中值滤波图像



(b)大小为 5×5 的中值滤波图像



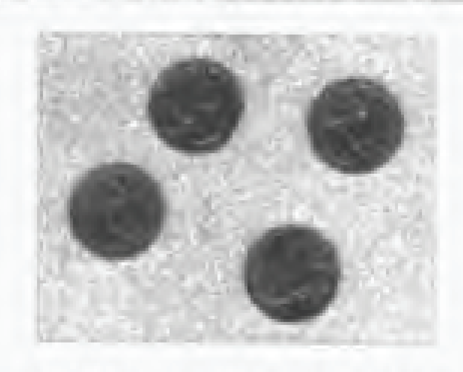
(c)大小为 7×7 的中值滤波图像

图 3-18

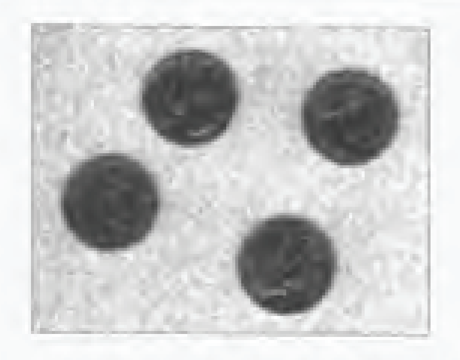
【例 3-7】采用 MATLAB 中的函数 filter2 对图像 3-17(b)进行 4 邻域和 8 邻域的平均滤波。滤波结果分别如图 3-19(a)和 3-19(b)所示。

```
[I,map]=imread('eight.tif');
J1=imnoise(I,'gaussian',0,0.02); % 受高斯噪声干扰，结果如图3-17(b)所示
M4=[0 1 0; 1 0 1; 0 1 0];
M4=M4/4;                     % 4邻域平均滤波
I_filter1=filter2(M4,J1);
figure,imshow(I_filter1,map);  % 结果如图3-19(a)所示

M8=[1 1 1; 1 0 1; 1 1 1];        % 8邻域平均滤波
M8=M8/8;
I_filter2=filter2(M8,J1);
figure,imshow(I_filter2,map);  % 结果如图3-19(b)所示
```



(a) 高斯噪声干扰的 4 邻域平均滤波



(b) 高斯噪声干扰的 8 邻域平均滤波

图 3-19

【例 3-8】图像的自适应魏纳滤波

MATLAB 中，可以利用 wiener2 函数对一幅图像进行自适应魏纳滤波。wiener2 函数的调

用格式如下：

```
[J,Noise] = wiener2(I,[m n])
```

其中，I表示输入的受噪声干扰的图像，[m,n]表示卷积使用的邻域大小，默认值为[3 3]，Noise是魏纳滤波器估计的噪声。

利用自适应魏纳滤波器对受高斯噪声干扰的图像3-17(b)进行滤波，滤波结果如图3-20所示。

```
[K noise]=wiener2(J1, [5 5]);
figure,imshow(K);                % 结果如图3-20所示
```

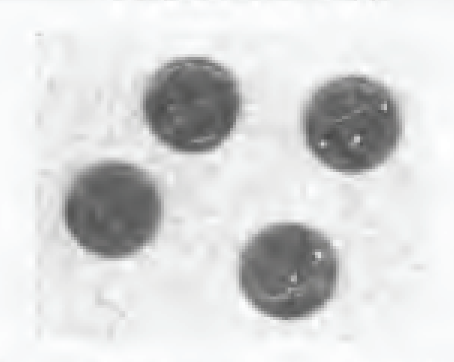


图3-20 高斯噪声干扰下的自适应魏纳滤波图像

3.4.3 空间域图像锐化

边缘模糊是图像中常出现的质量问题，由此造成的轮廓不清晰，线条不鲜明使图像特征提取、识别和理解难以进行。增强图像边缘和线条，使图像边缘变得清晰的处理称为图像锐化。本节讨论微分法、空间域高通滤波法和掩模法的图像锐化技术。

1. 微分法

图像模糊的实质是图像受到平均或积分运算，为实现图像的锐化，必须对它进行反运算"微分"，微分运算是求信号的变化率，有加强高频分量的作用，从而使图像轮廓清晰。为了把图像中间任何方向伸展的模糊边缘和轮廓变清晰，希望对图像的某种导数运算是各向同性的，可以证明梯度运算符合上述条件。

图像锐化中最常用的方法就是梯度法。设图像函数为$f(x,y)$，它的梯度是一个向量，定义：

$$\overline{G}[f(x,y)]=\begin{bmatrix}\dfrac{\partial f}{\partial x}\\ \dfrac{\partial f}{\partial y}\end{bmatrix} \tag{3-49}$$

在(x,y)点处的梯度，其方向是函数$f(x,y)$在这点变化率最大的方向，而其长度用$G[f(x,y)]$)表示，并用下式计算，即：

$$G[f(x,y)]=\sqrt{(\frac{\partial f}{\partial x})^2+(\frac{\partial f}{\partial y})^2} \tag{3-50}$$

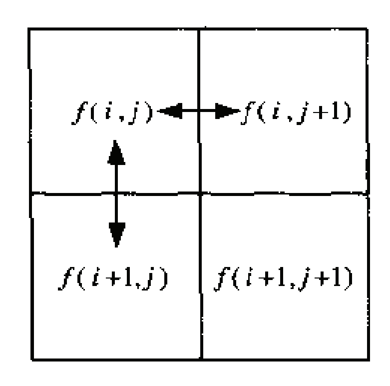
对于数字图像来说，则用差分来近似微分。常用的差分算法有两种：一种是典型的水平垂直差分算法，式（3-50）可以表示为：

$$G[f(x,y)]=\{[f(i,j)-f(i+1,j)]^2+[f(i,j)-f(i,j+1)]^2\}^{1/2} \quad (3\text{-}51)$$

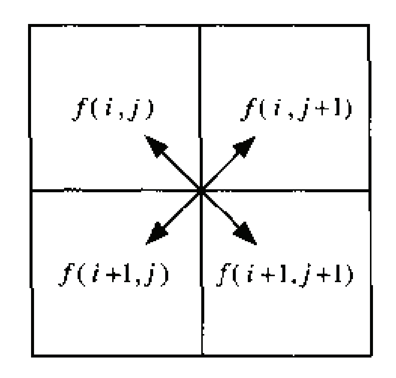
由于式(3-51)在计算机上运行时耗费时间很长，因此，实际应用中采用绝对查算法，(3-51)式也可以简化为：

$$G[f(x,y)]=|f(i,j)-f(i+1,j)|+|f(i,j)-f(i,j+1)| \quad (3\text{-}52)$$

式中各像素的位置关系如图 3-21(a)所示。



(a) 水平垂直差分算法



(b) 交叉垂直差分算法

图 3-21

另一种梯度法称为罗伯特梯度法（Robert Gradient），是一种交叉查分算法，如图 3-21(b)所示：

$$G[f(x,y)]=\{[f(i,j)-f(i+1,j+1)]^2+[f(i+1,j)-f(i,j+1)]^2\}^{1/2} \quad (3\text{-}53)$$

近似表达式为：

$$G[f(x,y)]=|f(i,j)-f(i+1,j+1)|+|f(i+1,j)-f(i,j+1)| \quad (3\text{-}54)$$

必须指出，对于一幅 $N\times M$ 数字图像，不可能在最后一行（$y=N$）和最后一列（$x=M$）像素上计算梯度值。实际应用中，用倒数第二行（$y=N-1$）和导数第二列（$x=M-1$）的梯度值近似代替。

梯度的计算表明，在图像中灰度变化较大的边沿区域其梯度值大，在灰度变化平缓的区域其梯度小，而在梯度均匀区域其梯度值接近于 0。

当梯度计算完后，根据实际需要采用不同的方法增强图像。第一种方法是使锐化后图像的各点灰度值 $g(x,y)$等于原始图像的在该点的梯度幅度，即：

$$g(x,y)=G[f(x,y)] \quad (3\text{-}55)$$

此法的缺点是增强后的图像仅显示灰度变化比较剧烈的边缘轮廓，而灰度变化平缓的区域则呈现一片黑色，大量信息丢失。

第二种增强图像是使

$$g(x,y)=\begin{cases}G[f(x,y)] & G[f(x,y)]\geq T\\ f(x,y) & \text{其他}\end{cases} \quad (3\text{-}56)$$

式中 T 是一个非负的阈值，适当选择 T，既可以明显突出图像中的边缘轮廓，而又不会破坏原始图像中灰度变化平缓的区域。

第三种增强图像是使：

$$g(x,y)=\begin{cases}L_G & G[f(x,y)]\geqslant T \\ f(x,y) & \text{其他}\end{cases} \tag{3-57}$$

式中 L_G 是指定的一个灰度级，它将明显的边缘用固定的灰度级 L_G 来表征。

第四种增强图像是：

$$g(x,y)=\begin{cases}G[f(x,y)] & G[f(x,y)]\geqslant T \\ L_B & \text{其他}\end{cases} \tag{3-58}$$

将背景用一个固定的灰度级 L_B 来实现，便于研究边缘处的灰度变化。

最后，如果对目标的边缘位置感兴趣，则可利用下式增强图像：

$$g(x,y)=\begin{cases}L_G & G[f(x,y)]\geqslant T \\ L_B & \text{其他}\end{cases} \tag{3-59}$$

这一算法给出二值图像，便于研究边缘所在位置。该算法实质与梯度边缘检测相类似，有关内容将在下一章详细介绍。

【例 3-9】分别运用 5 种不同的梯度增强法对图 3-22(a)进行图像锐化，其锐化结果分别如图 3-22(b)～(f)所示。

```
[I,map]=imread('3-22.jpg');
imshow(I,map);
I=double(I);
[Gx,Gy]=gradient(I);        % 计算梯度
G=sqrt(Gx.*Gx+Gy.*Gy);    % 注意是矩阵点乘

J1=G;
figure,imshow(J1,map);    % 第一种图像增强

J2=I;                     % 第二种图像增强
K=find(G>=7);
J2(K)=G(K);
figure,imshow(J2,map);

J3=I;                     % 第三种图像增强
K=find(G>=7);
J3(K)=255;
figure,imshow(J3,map);

J4=I;                     % 第四种图像增强
K=find(G<=7);
J4(K)=255;
figure,imshow(J4,map);
```

```
J5=I;                           % 第五种图像增强
K=find(G<=7);
J5(K)=0;
Q=find(G>=7);
J5(Q)=255;
figure,imshow(J5,map);
```



(a) 梯度图像锐化目标图像



(b) 第一种图像锐化效果



(c) 第二种图像锐化效果



(d) 第三种图像锐化效果



(e) 第四种图像锐化效果



(f) 第五种图像锐化效果

图 3-22

2. 高通滤波

图像中的边缘与图像频谱中的高频分量相对应，因此采用高通滤波器让高频分量顺利通过，而对低频分量则充分限制，使图像的边缘变得清晰，实现图像的锐化。空间域高通滤波建立在离散卷积的基础上，即：

$$g(x,y)=\sum_{(i,j)}\sum_{\in S} f(i,j)H(x-i+1,y-j+1) \tag{3-60}$$

式中 $g(x,y)$为锐化输出，$f(i,j)$为输入图像，$H(x\text{-}i\text{+}1,n\text{-}j\text{+}1)$为冲激响应阵列（或卷积阵列）。常用阵列有以下几种：

$$H_1=\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix} \qquad H_2=\begin{bmatrix}-1 & -1 & -1\\ -1 & 9 & -1\\ -1 & -1 & -1\end{bmatrix} \qquad H_3=\begin{bmatrix}1 & -2 & 1\\ -2 & 5 & -2\\ 1 & -2 & 1\end{bmatrix}$$

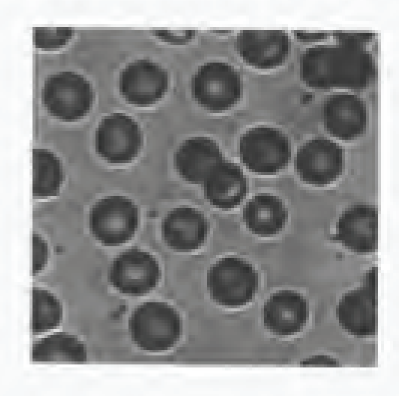
3. 掩模法

高通滤波的效果也可以用原始图像减去低通图像得到。如果原始图像乘以一个放大系数，然后再减去低通图像可以构成一幅高频增强图像，这样的图像恢复了部分高通滤波时丢失的低频成分，使得最终结果与原始图像更为接近，这种操作也称为（非锐化）掩模。

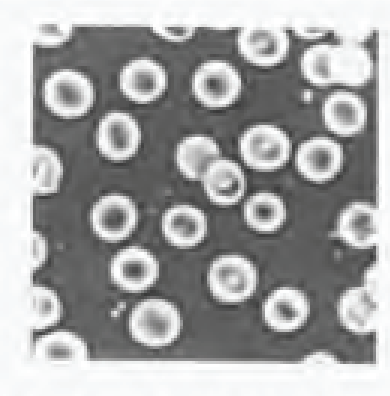
【例 3-10】对图 3-23(a)用 H_2 模板进行高通滤波和掩模处理，经处理后的图像分别如图 3-23(b)和 3-23(c)所示。

```
[I,map]=imread('blood1.tif');
imshow(I,map);
H2=[-1 -1 -1;-1 9 -1;-1 -1 -1];
J1=filter2(H2,I);                 % 高通滤波
figure,imshow(J1,map);

I=double(I);
M=[1 1 1;1 1 1;1 1 1]/9;
J2=filter2(M,I);
J3=I-J2;                % 掩模
figure,imshow(J3,map);
```



(a) 原始图像



(b) H_2 模板高通滤波处理的图像



(c) 掩模法处理后的图像

图 3-23

3.5 频域增强

3.5.1 频域低通滤波

一般来说，图像的边缘和噪声都对应于傅立叶变换中的高频分量，所以通过频域对一定范围的高频分量的衰减能够达到图像平滑、去除噪声。

频域滤波可用下述关系式表示：

$$G(u,v)=H(u,v)F(u,v) \tag{3-61}$$

式中 $F(u,v)$是需要平滑处理的图像$f(x,y)$的变换。通过函数 $H(u,v)$使 $F(u,v)$的高频分量衰减，输出 $G(u,v)$，$G(u,v)$经反变换得到平滑图像 $g(x,y)$。由于滤除了高频分量，低频信息无损地通过，所以称为低通滤波。函数 $H(u,v)$ 称为滤波器传递函数。常用的几种频域低通滤波器有理想低通滤波器、巴特沃斯（Butterworth）低通滤波器和指数低通滤波器。

1. 理想低通滤波器

一个二维的理想低通滤波器（ILPF）的传递函数为：

$$H(u,v)=\begin{cases}1 & D(u,v)\leqslant D_0\\ 0 & D(u,v)>D_0\end{cases} \tag{3-62}$$

式中 D 是从点（x,y）到频率平面地原点距离，即：

$$D(u,v)=\sqrt{u^2+v^2} \tag{3-63}$$

D_0表示截止频率点到原点地距离。理想低通滤波器的特性曲线如图 3-24 所示。它表示以 D_0 为半径的圆内的所有频率成分无损地通过，圆外地所有频率分量完全衰减。但必须指出，在 MATLAB 仿真环境下，可以模拟一个理想低通滤波器，实际应用中是无法用电子元件设计出特性曲线如图 3-24 的理想滤波器的。

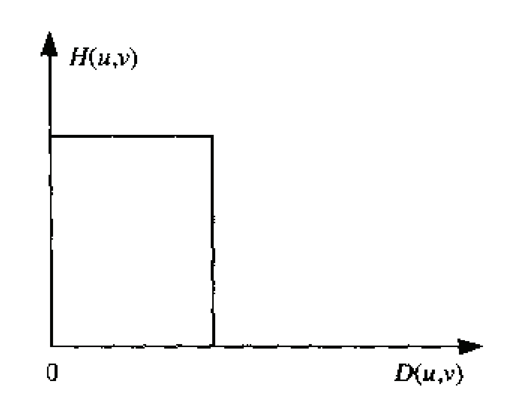


图 3-24 理想低通滤波器的特性曲线

2. 巴特沃斯（Butterworth）低通滤波器

具有从原点开始到距离 D_0 处的截止频率轨迹的 n 阶巴特沃思低通滤波器（BLPF）的传递函数可以由下式决定：

$$H(u,v)=\frac{1}{1+\left[\dfrac{D(u,v)}{D_0}\right]^{2n}} \tag{3-64}$$

巴特沃思低通滤波函数的特性曲线示如图 3-25 所示。

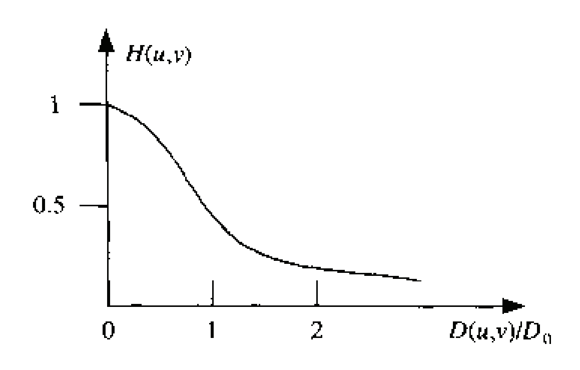


图 3-25　巴特沃思低通滤特性曲

不同于理想低通滤波器，巴特沃思低通滤波器传递函数在通过频率与滤去频率之间没有明显的不连续性。对具有平滑传递函数的滤波器通常是将 $H(u,v)$下降到最大值的某一分数值的那个点，定义为截止频率。通常用的一个值是等于 $H(u,v)$最大值的$\sqrt{2}/2$。则（3-64）式可以简化为：

$$H(u,v)=\frac{1}{1+(\sqrt{2}-1)\left[\dfrac{D(u,v)}{D_0}\right]^{2n}} \tag{3-65}$$

3. 指数滤波器

指数低通滤波器（ELPF）是在图像处理中常用的另一种平滑滤波器。具有截止频率 D_0 的指数低通滤波器，它的传递函数为，

$$H(u,v)e^{-[D(u,v)/D_0]^n} \tag{3-66}$$

式中 n 决定指数函数的衰减率。指数低通滤波器的特性曲线如图 3-26 所示。通常选用 $H(u,v)$最大值的$\sqrt{2}/2$时的频率作为高通滤波器的截止频率，则（3-66）式简化为：

$$H(u,v)=e^{\ln(1/\sqrt{2})[D(u,v)/D_0]^n} \tag{3-67}$$

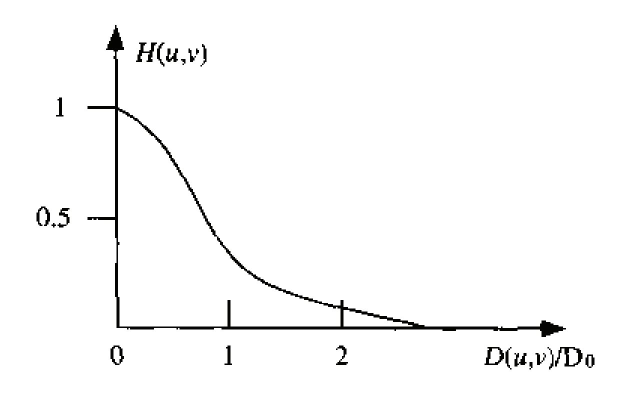


图 3-26　指数低通滤波器特性曲线

4. 梯形低通滤波器

梯形低通滤波器是理想低通滤波器和完全平滑滤波器之间的折衷，它的传递函数表示为：

$$H(u,v)=\begin{cases}1 & D(u,v)<D_0\\ \dfrac{1}{(D_0-D_1)}[D(u,v)-D_1] & D_0\leqslant D(u,v)\leqslant D_1\\ 0 & D(u,v)>D_1\end{cases} \tag{3-68}$$

式中 D_0 和 D_1 是指定的，且 $D_0<D_1$。梯形滤波器的传递函数特性曲线如图 3-27 所示。实际应用中，为简单起见，将 D_0 定义为截止频率。

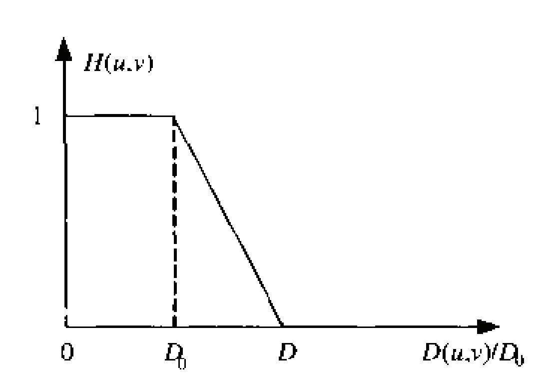
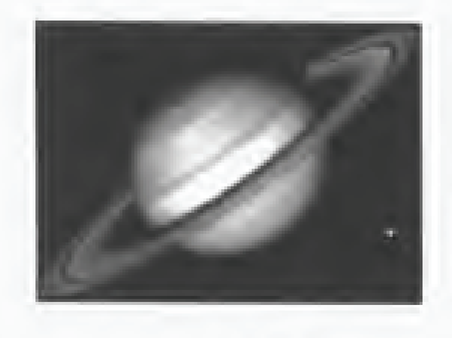


图 3-27 梯形滤波器的特性曲线

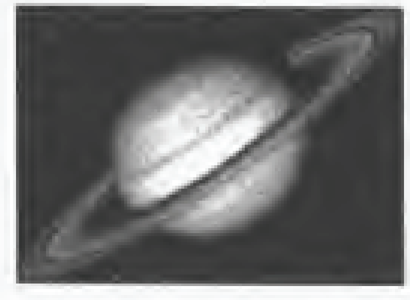
【例 3-11】对图 3-28(a)受椒盐噪声干扰后的图像如图 3-28(b)所示，利用巴特沃斯（Butterworth）低通滤波器进行平滑处理。滤波后的图像如图 3-28(c)所示。

```
I=imread('Saturn.tif');
imshow(I);
J1=imnoise(I,'salt & pepper');   % 叠加椒盐噪声
figure,imshow(J1);
f=double(J1);        % 数据类型转换，MATLAB 不支持图像的无符号整型的计算
g=fft2(f);           % 傅立叶变换
g=fftshift(g);        % 转换数据矩阵
[M,N]=size(g);
nn=2;               % 二阶巴特沃斯(Butterworth)低通滤波器
d0=50;
m=fix(M/2); n=fix(N/2);
for i=1:M
     for j=1:N
         d=sqrt((i-m)^2+(j-n)^2);
         h=1/(1+0.414*(d/d0)^(2*nn));   % 计算低通滤波器传递函数
         result(i,j)=h*g(i,j);
     end
end
result=ifftshift(result);
J2=ifft2(result);
J3=uint8(real(J2));
```

```
figure,imshow(J3);                          % 显示滤波处理后的图像
```



(a)原始图像



(b)受椒盐噪声干扰的图像



(c)二阶 Butterworth 低通滤波去噪图像

图 3-28

3.5.2 频域高通滤波

由于图像中灰度发生骤变的部分与其频谱的高频分量相对应，所以采用高通滤波器衰减或抑制低频分量，使高频分量畅通并能够对图像进行锐化处理。常用的几种频域高通滤波器有理想高通滤波器、巴特沃斯（Butterworth）高通滤波器和指数高通滤波器。

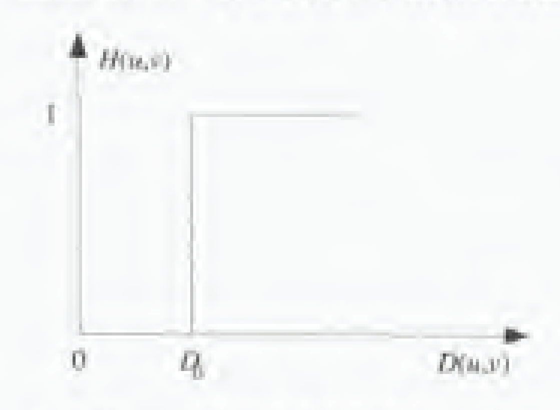


图 3-29 二维理想高通滤波器的特性曲线

二维理想高通滤波器的传递函数为：

$$H(u,v)=\begin{cases}0 & D(u,v)\leqslant D_0\\1 & D(u,v)>D_0\end{cases} \tag{3-69}$$

传递函数的特性曲线如图 3-29 所示。

巴特沃思高通滤波器的传递函数为：

$$H(u,v)=\frac{1}{1+\left[\dfrac{D_0}{D(u,v)}\right]^{2n}} \tag{3-70}$$

传递函数的特性曲线如图 3-30 所示。通常选用 $H(u,v)$ 最大值的 $\sqrt{2}/2$ 时的频率作为高通滤波器的截止频率，则（3-68）式简化为：

$$H(u,v)=\frac{1}{1+(\sqrt{2}-1)\left[\dfrac{D_0}{D(u,v)}\right]^{2n}} \tag{3-71}$$

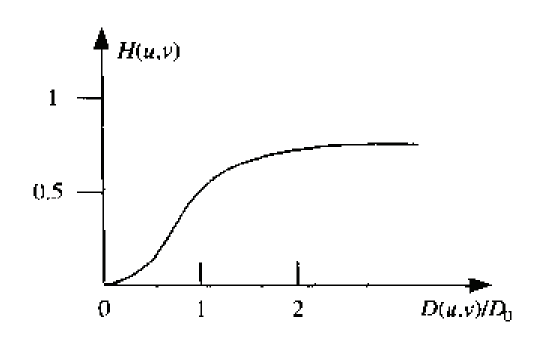


图 3-30　巴特沃思高通滤波器的特性曲线

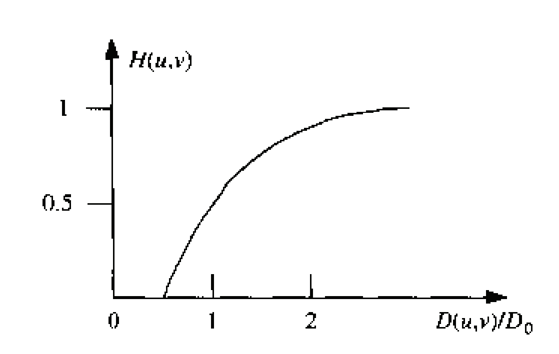


图 3-31　指数滤高通波器的特性曲线

指数滤波器的传递函数为：

$$H(u,v)=e^{-[D_0/D(u,v)]^n} \tag{3-72}$$

传递函数的特性曲线如图 3-31 所示。通常选用 H(u,v)最大值的 $\sqrt{2}/2$ 时的频率作为高通滤波器的截止频率，则（3-71）式简化为，

$$H(u,v)=e^{\ln(1/\sqrt{2})[D_0/D(u,v)]^n} \tag{3-73}$$

梯形高通滤波器的传递函数为：

$$H(u,v)=\begin{cases}0 & D(u,v)<D_0\\ \dfrac{1}{(D_0-D_1)}[D(u,v)-D_1] & D_0\leqslant D(u,v)\leqslant D_1\\ 1 & D(u,v)>D_1\end{cases} \tag{3-74}$$

传递函数的特性曲线如图 3-32 所示。

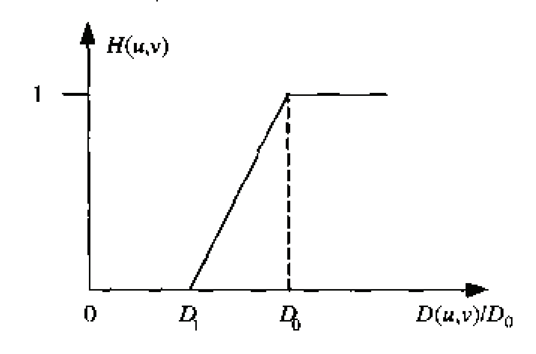
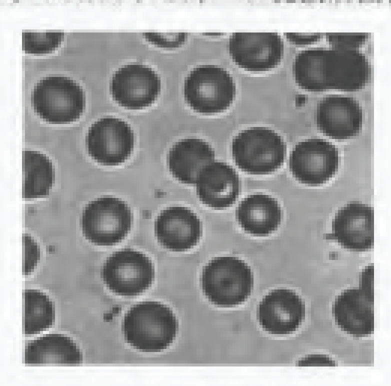


图 3-32　梯形高通滤波器的特性曲线

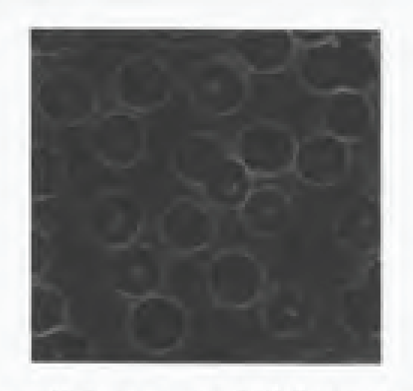
【例 3-12】对图像 3-33(a)利用巴特沃斯（Butterworth）高通滤波器进行锐化处理。处理结果如图 3-33(b)所示。

```
I=imread('blood1.tif');
imshow(I);
```

```
f=double(I);     % 数据类型转换，MATLAB 不支持图像的无符号整型的计算
g=fft2(f);        % 傅立叶变换
g=fftshift(g);    % 转换数据矩阵
[M,N]=size(g);
nn=2;           % 二阶巴特沃斯(Butterworth)高通滤波器
d0=5;
m=fix(M/2);
n=fix(N/2);
for i=1:M
      for j=1:N
          d=sqrt((i-m)^2+(j-n)^2);
          if (d==0)
             h=0;
          else
             h=1/(1+0.414*(d0/d)^(2*nn));
          end
result(i,j)=h*g(i,j);
end
end
result=ifftshift(result);
J2=ifft2(result);
J3=uint8(real(J2));
figure,imshow(J3);  % 滤波后图像显示
```



(a) 原始图像



(b) 二阶 Butterworth 高通滤波图像

图 3-33

第 4 章　图像分割

图像分割（Image Segmentation）是把图像分割成若干个特定的、具有独特性质的区域并提取出感兴趣目标的技术和过程。在对图像的研究和应用中，人们往往仅对图像的某些部分感兴趣（目标或背景），它们一般对应图像中特定的、具有独特性质的区域。为了分析和识别目标，需要将它们分割并提取出来。

图像分割是由图像处理转到图像分析的关键。一方面，它是目标图像表达的基础，对特征测量有重要的影响。另一方面，图像分割和分割的目标表达、特征提取和参数测量等将原始图像转化为数学表达形式，使得利用计算机进行图像分析和理解成为可能。本章主要介绍图像分割的基本概念和分割所用的主要方法。

4.1　图像分割的基本概念

4.1.1　图像分割定义

有关图像分割的解释和表述很多，借助于集合概念对图像分割可给出如下比较正式的定义：

令集合 R 代表整幅图像的区域，对 R 的分割可看成将 R 分成 N 个满足以下 5 个条件的非空子集（子区域）$R_1,R_2,\cdots,R_N$：

① $\bigcup_{i=1}^{N} R_i = R$；

② 对所有的 i 和 j，有 $i \neq j$，$R_i \cap R_j = \varnothing$；

③ 对 $i = 1,2,\cdots,N$，有 $P(R_i) = \text{True}$；

④ 对 $i \neq j$，$P(R_i \cup R_j) = \text{False}$；

⑤ $i = 1,2,\cdots,N$，R_i 是连通的区域。

其中 $P(R_i)$ 是对所有在集合中元素的逻辑谓词，$\varnothing$ 代表空集。

下面先对上述各个条件分别给予简略解释。条件①指出对一幅图像分割所得到的全部子区域的总和（并集）应能包括图像中所有像素（就是原图像），或者说分割应将图像中的每个像素都分进某个子区域中。条件②指出在分割结果中各个子区域是互不重叠的，或者说在分割结果中一个像素不能同时属于两个区域。条件③指出在分割结果中每个子区域都有独特的特性，

或者说属于同一个区域中的像素应该具有某些相同特性。条件④指出在分割结果中，不同的子区域具有不同的特性，没有公共元素，或者说属于不同区域的像素应该具有一些不同的特性。条件⑤要求分割结果中同一个子区域内的像素应当是连通的，即同一个子区域内的任两个像素在该子区域内互相连通、或者说分割得到的区域是一个连通组元。

另外，上述这些条件不仅定义了分割，也对进行分割有指导作用。对图像的分割总是根据一些分割准则进行的。条件①与条件②说明正确的分割准则应可适用于所有区域和所有像素，而条件③与条件④说明合理的分割准则应能帮助确定各区域像素有代表性的特性，条件⑤说明完整的分割准则应直接或间接地对区域内像素的连通性有一定的要求或限定。

最后需要指出，实际应用中图像分割不仅要把一幅图像分成满足上面 5 个条件的各具特性的区域而且需要把其中感兴趣的目标区域提取出来。只有这样才算真正完成了图像分割的任务。本章所讨论的图像分割如没有特殊说明都这样来考虑。

4.1.2 图像分割算法分类

图像分割算法的研究一直受到人们的高度重视，到目前为止，提出的分割算法已经多达上千种，由于现有的分割算法非常多，所以将它们进行分类的方法也提出了不少。例如有把分割算法分成 3 类的：①边缘检测、②阈值分割和③区域生长。但事实上阈值分割的方法在本质上也是一种区域提取方法，所以③实际上包含了①。另外也有把分割算法分成六部分讨论的：①阈值分割、②像素分类、③深度图像分割、④彩色图像分割、⑤边缘检测和⑥基于模糊集的方法。从算法的角度来看，各部分内容是有重叠的。事实上对深度图像和彩色图像的分割仍须用或可用①、②或⑤这几部分中的方法进行，而⑥所讨论的只是把模糊集合理论用于①、②和⑤中的方法里。另外①和②中的方法有许多相似之处，而常见的基于区域生长原理的算法却没有包含进这些类中去。

本章从实际应用的角度考虑，详细介绍了图像分割的如下算法：边缘检测、阈值分割、区域生长、彩色图像分割和特殊理论工具的分割等 5 部分。

4.2 边缘检测

4.2.1 边缘检测概述

边缘（Edge）是指图像局部亮度变化最显著的部分。边缘主要存在于目标与目标、目标与背景、区域与区域（包括不同色彩）之间，是图像分割、纹理特征提取和形状特征提取等图像分析的重要基础。图像分析和理解的第一步常常是边缘检测（Edge Detection）。由于边缘检测十分重要，因此成为机器视觉研究领域最活跃的课题之一。本节主要讨论边缘检测和定位的基本概念，并通过几种常用的边缘检测器来说明边缘检测的基本问题。

图像中的边缘通常与图像亮度或图像亮度的一阶导数的不连续性有关。图像亮度的不连续可分为：①阶跃不连续，即图像亮度在不连续处的两边的像素灰度值有着显著的差异；②线条不连续，即图像亮度突然从一个值变化到另一个值，保持一个较小的行程后又返回到原来的值。在实际中，阶跃和线条边缘图像是很少见的，由于大多数传感元件具有低频特性，使得阶跃边

缘变成斜坡型边缘，线条边缘变成屋顶形边缘，其中的亮度变化不是瞬间的，而是跨越一定的距离。

对一个边缘来说，有可能同时具有阶跃和线条边缘特性，例如在一个表面上，由一个平面变化到法线方向不同的另一个平面就会产生阶跃边缘；如果这一表面具有镜面反射特性且两平面形成的棱角比较圆滑，则当棱角圆滑表面的法线经过镜面反射角时，由于镜面反射分量，在棱角圆滑表面上会产生明亮光条，这样的边缘看起来像在阶跃边缘上叠加了一个线条边缘。由于边缘可能与场景中物体的重要特征对应，所以它是很重要的图像特征。比如，一个物体的轮廓通常产生阶跃边缘，因为物体的图像亮度不同于背景的图像亮度。

在讨论边缘算子之前，首先给出下列术语的定义。

边缘点：图像中亮度显著变化的点。

边缘段：边缘点坐标$[i,j]$及其方向 θ 的总和，边缘的方向可以是梯度角。

边缘检测器：从图像中抽取边缘（边缘点或边缘段）集合的算法。

轮廓：边缘列表，或是一条边缘列表的曲线模型。

边缘连接：从无序边缘表形成有序边缘表的过程。习惯上边缘的表示采用顺时针方向来排序。

边缘跟踪：一个用来确定轮廓图像（指滤波后的图像）的搜索过程。

边缘点的坐标可以是边缘位置像素点的行、列整数标号，也可以在子像素分辨率水平上表示。边缘坐标可以在原始图像坐标系上表示，但大多数情况下是在边缘检测滤波器的输出图像的坐标系上表示，因为滤波过程可能导致图像坐标平移或缩放，边缘段可以用像素点尺寸大小的小线段定义，或用具有方向属性的一个点定义。请注意，在实际应用中，边缘点和边缘段都称为边缘。

由于边缘检测器生成的边缘集分成两个：真边缘和假边缘集。真边缘集对应场景中的边缘，假边缘集不是场景中的边缘。还有一个边缘集，即场景中的漏检边缘集。假边缘集称之为假阳性（False Positive），而漏掉的边缘集则称之为假阴性（False Negative）。

边缘连接和边缘跟踪之间的区别在于：边缘连接是把边缘检测器产生的无序边缘集作为输入，输出一个有序边缘集；边缘跟踪则是将一幅图像作为输入，输出一个有序边缘集。另外，边缘检测使用局部信息来决定边缘，而边缘跟踪使用整个图像信息来决定一个像素点是不是边缘。

4.2.2 边缘检测梯度算法

1. 梯度边缘检测算法基本步骤

梯度边缘检测算法有如下 4 个步骤。

（1）滤波：边缘检测算法主要是基于图像强度的一阶和二阶导数，但导数的计算对噪声敏感，因此必须使用滤波器来改善与噪声有关的边缘检测器的性能。需要指出，大多数滤波器在降低噪声的同时也导致了边缘强度的损失，因此，增强边缘和降低噪声之间需要折衷。

（2）增强：增强边缘的基础是确定图像各点邻域强度的变化值。增强算法可以将邻域（或局部）强度值有显著变化的点突显出来。边缘增强一般是通过计算梯度幅值来完成的。

（3）检测：在图像中有许多点的梯度幅值比较大，而这些点在特定的应用领域中并不都是边缘，所以应该用某种方法来确定哪些点是边缘点。最简单的边缘检测判据是梯度幅值阈值判据。

（4）定位：如果某一应用场合要求确定边缘位置，则边缘的位置可在子像素分辨率上来

估计，边缘的方位也可以被估计出来。

在边缘检测算法中，前 3 个步骤用得十分普遍。这是因为大多数场合下，仅仅需要边缘检测器指出边缘出现在图像某一像素点的附近，而没有必要指出边缘的精确位置或方向。边缘检测误差通常是指边缘误分类误差，即把假边缘判别成边缘而保留，而把真边缘判别成假边缘而去掉。边缘估计误差是用概率统计模型来描述边缘的位置和方向误差的。我们将边缘检测误差和边缘估计误差区分开，是因为它们的计算方法完全不同，其误差模型也完全不同。

人们已经发展了许多边缘检测器，这里仅讨论常用的几种边缘检测器。

2. Roberts 算子

Roberts 交叉算子为梯度幅值计算提供了一种简单的近似方法：

$$G(i,j)=|f(i,j)-f(i+1,j+1)|+|f(i+1,j)-f(i,j+1)| \tag{4-1}$$

用卷积模板表示方法，上式变成：

$$G(i,j)=|G_x|+|G_y| \tag{4-2}$$

其中 G_x 和 G_y 由下面的模板计算：

$$G_x=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \qquad G_y=\begin{bmatrix}0 & -1\\ 1 & 0\end{bmatrix} \tag{4-3}$$

在计算梯度时，计算空间同一位置处(x,y)的真实偏导数非常重要。采用上述 2×2 邻域模板计算的梯度近似值 G_x 和 G_y 并不位于同一位置，G_x 实际上是内差点（$i,j+1/2$）处的近似梯度，G_y 实际上是内差点（$i+1/2,j$）处的近似梯度。因此，Roberts 算子是该点连续梯度的近似值，而不是所预期点（i,j）处的近似值。所以，通常用 3×3 邻域计算梯度值。

3. Sobel 算子

正如前面所讲，采用 3×3 邻域可以避免在像素之间内插点上计算梯度。考虑下图中所示的点（i,j）周围点的排列。

a_0	a_1	a_2
a_7	(i,j)	a_3
a_6	a_5	a_4

Sobel 算子也是一种梯度幅值：

$$M=\sqrt{s_x^2+s_y^2} \tag{4-4}$$

其中的偏导数用下式计算：

$$s_x=(a_2+ca_3+a_4)-(a_0+ca_7+a_6) \tag{4-5}$$

$$s_y = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6) \tag{4-6}$$

其中常系数 c=2。

和其他的梯度算子一样，s_x 和 s_y 可分别用卷积模板表示为：

−1	0	1
−2	0	2
−1	0	1

1	2	1
0	0	0
−1	-2	-1

这一算子把重点放在接近于模板中心的像素点。Sobel 算子是边缘检测器中最常用的算子之一。

4. Prewitt 算子

Prewitt 算子与 Sobel 算子的方程完全一样，只是常系数 c=1。所以 s_x 和 s_y 可分别用卷积模板表示为：

−1	0	1
−1	0	1
−1	0	1

1	1	1
0	0	0
−1	-1	-1

5. 边缘检测实例

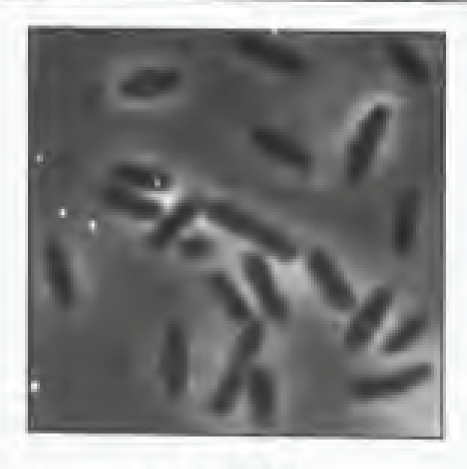
可以使用 MATLAB 图像处理工具箱中的 edge 函数利用以上算子来检测边缘。edge 函数提供许多微分算子模板，对于某些模板可以指定其是对水平边缘还是对垂直边缘（或者二者都有）敏感（即主要检测是水平边缘还是垂直边缘）。edge 函数在检测边缘时可以指定一个灰度阈值，只有满足这个阈值条件的点才视为边界点。edge 函数的基本调用格式如下：

$$BW = \text{edge}(I, '\mathit{type}', \mathit{parameter}, \cdots)$$

其中，I 表示输入图像，*type* 表示使用的算子类型，*parameter* 则是与具体算子有关的参数。

【例 4-1】用 Prewitt 算子检测图像 4-1(a)的边缘，检测结果如图 4-1(b)所示。以下语句将用 Prewitt 算子检测边缘：

```
I = imread('bacteria.BMP');
BW1 = edge(I,'prewitt',0.04);          % 0.04 为梯度阈值
figure(1);
imshow(I);
figure(2);
imshow(BW1);
```



(a) 原始图像



(b) Prewitt 算子检测的边缘

图 4-1

4.2.3 拉普拉斯（Laplacian）算子

前面讨论了计算一阶导数的边缘检测器，如果所求的一阶导数高于某一阈值，则可确定该点为边缘点。这样做会导致检测的边缘点太多。一种更好的方法就是求梯度局部最大值对应的点，并认定它们是边缘点，如图 4-2 所示。在图 4-2 中，若用阈值来进行边缘检测，则在 a 和 b 之间的所有点都被记为边缘点。但通过去除一阶导数中的非局部最大值，可以检测出更精确的边缘。一阶导数的局部最大值对应着二阶导数的零交叉点（Zero crossing）。这样，通过找图像强度的二阶导数的零交叉点就能找到精确边缘点。拉普拉斯（Laplacian）算子是常用的二阶导数算子。

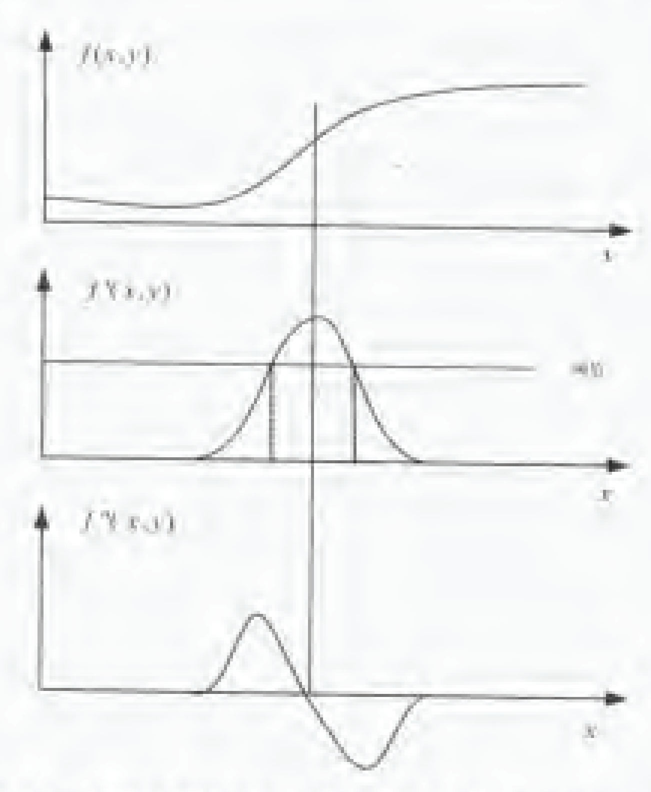


图 4-2　阈值边缘检测和二阶导数的零交叉边缘检测比较

平滑过的阶跃边缘二阶导数是一个在边缘点处过零的函数（见图 4-2)。拉普拉斯算子是二阶导数的二维等效式，函数 $f(x,y)$的拉普拉斯算子公式为：

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{4-7}$$

使用差分方程对 x 和 y 方向上的二阶偏导数近似如下：

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial G_x}{\partial x} = \frac{\partial(f(i,j+1)-f(i,j))}{\partial x} = \frac{\partial f(i,j+1)}{\partial x} - \frac{\partial f(i,j)}{\partial x}$$

$$= f(i,j+2) - 2f(i,j+1) + f(i,j) \tag{4-8}$$

上式近似是以点（$i,j+1$）为中心的，以点（i,j）为中心的近似式为：

$$\frac{\partial^2 f}{\partial x^2} = f(i,j+1) - 2f(i,j) + f(i,j-1) \tag{4-9}$$

类似地有：

$$\frac{\partial^2 f}{\partial y^2} = f(i+1,j) - 2f(i,j) + f(i-1,j) \tag{4-10}$$

将（4-9）和（4-10）两式合并为一个算子，用近似的拉普拉斯算子模板表示：

$$\nabla^2 \approx \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4-11}$$

当拉普拉斯算子输出出现过零点时就表明有边缘存在，但是要去除无意义的零点（灰度值为 0 的区域)。

4.2.4 LoG（Laplacian-Gauss）算子

前面介绍的梯度算子和拉普拉斯算子实质都是微分或差分算法，因此算法对噪声十分敏感。所以，在边缘检测前，必须滤除噪声。Marr 和 Hildreth 将高斯滤波和拉普拉斯边缘检测结合在一起，形成 LoG（Laplacian-Gauss）算法。LoG 边缘检测器的基本特征是：

（1）平滑滤波器是高斯滤波器；

（2）增强步骤采用二阶导数（二维拉普拉斯函数)；

（3）边缘检测判据是二阶导数零交叉点并对应一阶导数的较大峰值；

（4）使用线性内插方法在子像素分辨率水平上估计边缘的位置。

这种方法的特点是图像首先与高斯滤波器进行卷积，这一步既平滑了图像又降低了噪声，孤立的噪声点和较小的结构组织将被滤除。由于平滑会导致边缘的延展，因此边缘检测器只考虑那些具有局部梯度最大值的点为边缘点。这一点可以用二阶导数的零交叉点来实现。拉普拉斯函数用作二维二阶导数的近似，是因为它是一种无方向算子。为了避免检测出非显著边缘，应选择一阶导数大于某一阈值的零交叉点作为边缘点。

LoG 算子对图像 $f(x,y)$ 进行边缘检测，输出 $h(x,y)$ 是通过卷积运算得到的，即：

$$h(x,y)=\left[\left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}\right]*f(x,y) \tag{4-12}$$

滤波（或平滑）、增强和检测 3 个边缘检测的步骤对 LoG 算子边缘检测仍然成立。其中高斯滤波器对图像进行平滑，拉普拉斯算子将边缘点转换成零交叉点来实现，边缘检测通过零交叉点的检测实现。

前一章已经介绍，图像的平滑会引起边缘的模糊。高斯平滑运算导致图像中边缘和其他尖锐不连续部分的模糊，其中模糊量取决于σ值。σ值越大，噪声滤波效果越好，但同时也丢失了重要的边缘信息，影响了边缘检测器的性能。如果取小σ值，又有可能平滑不完全而留有太多的噪声。大σ值的滤波器在平滑相互邻近的两个边缘时，可能会将它们连在一起，这样只能检测出一个边缘。因此，在不知道物体尺度和位置的情况下，很难准确确定滤波器的σ值。一般来说，使用大σ值的滤波器产生鲁棒边缘，小σ值的滤波器产生精确定位的边缘，两者结合，能够检测出图像的最佳边缘。

LoG 算子也可以用 MATLAB 中的 edge 函数来检测边缘：

```
BW = edge(I,'log',thresh,sigma)
```

式中 thresh 是边缘检测的阈值，sigma 是高斯滤波器的σ值，默认为 2。

【例 4-2】用不同σ值的 LoG 算子检测图像 4-3(a)的边缘。检测结果分别如 4-3(a)和 4-3(b)所示。

```
I = imread('bacteria.BMP');
BW1 = edge(I,'log',0.003);  % σ=2
imshow(BW1); title('σ=2')
BW1 = edge(I,'log',0.003,3); % σ=3
figure, imshow(BW1); title('σ=3')
```



(a) σ=2 的 LoG 算子检测的边缘



(b) σ=3 的 LoG 算子检测的边缘



(c) 坎尼算子检测的边缘

图 4-3

比较上述两幅图像可以发现，σ值小，平滑程度轻，会出现零星的假边缘；σ值大，平滑程度重，但是部分真实的边缘丢失，出现边缘间断现象。

4.2.5 坎尼（Canny）算子

检测阶跃边缘的基本思想是在图像中找出具有局部最大梯度幅值的像素点。检测阶跃边缘的大部分工作集中在寻找能够用于实际图像的梯度数字逼近。由于实际的图像经过了摄像机光学系统和电路系统（带宽限制）固有的低通滤波器的平滑，因此，图像中的阶跃边缘不是十分陡立。图像也受到摄像机噪声和场景中不希望的细节的干扰。图像梯度逼近必须满足两个要求：①逼近必须能够抑制噪声效应；②必须尽量精确地确定边缘的位置。抑制噪声和边缘精确定位是无法同时得到满足的，也就是说，边缘检测算法通过图像平滑算子去除了噪声，但却增加了边缘定位的不确定性；反过来，若提高边缘检测算子对边缘的敏感性，同时也提高了对噪声的敏感性。有一种线性算子可以在抗噪声干扰和精确定位之间选择一个最佳折衷方案，它就是高斯函数的一阶导数，对应于图像的高斯函数平滑和梯度计算。

在高斯噪声中，一个典型的边缘代表一个阶跃的强度变化。根据这个模型，好的边缘检测算子应该有 3 个指标：①低失误概率，即真正的边缘点尽可能少的丢失又要尽可能避免将非边缘点检测为边缘；②高位置精度，检测的边缘应尽可能接近真实的边缘；③对每一个边缘点有惟一的响应，得到单像素宽度的边缘。坎尼算子提出了边缘检测算子的如下 3 个准则。

（1）信噪比准则

信噪比越大，提取的边缘质量越高。信噪比 SNR 定义为：

$$SNR=\frac{\left|\int_{-W}^{+W}G(-x)h(x)\mathrm{d}x\right|}{\sigma\sqrt{\int_{-W}^{+W}h^2(x)\mathrm{d}x}} \tag{4-13}$$

其中 $G(x)$代表边缘函数，$h(x)$代表宽度为 W 的滤波器的脉冲响应，σ代表高斯噪声的均方差。

（2）定位精度准则

边缘定位精度 L 如下定义：

$$L=\frac{\left|\int_{-W}^{+W}G'(-x)h'(x)\mathrm{d}x\right|}{\sigma\sqrt{\int_{-W}^{+W}h'^2(x)\mathrm{d}x}} \tag{4-14}$$

其中 $G'(x)$和 $h'(x)$分别是 $G(x)$和 $h(x)$的导数。L 越大表明定位精度越高。

（3）单边缘响应准则

为了保证单边缘只有一个响应，检测算子的脉冲响应导数的零交叉点平均距离 $D(f')$应满足：

$$D(f')=\pi\left\{\frac{\int_{-\infty}^{+\infty}h'^2(x)\mathrm{d}x}{\int_{-W}^{+W}h''(x)\mathrm{d}x}\right\}^{\frac{1}{2}} \tag{4-15}$$

$h''(x)$是 $h(x)$的二阶导数。

以上述指标和准则为基础，利用泛函求导的方法可导出坎尼边缘检测器是信噪比与定位之乘积的最优逼近算子，表达式近似于高斯函数的一阶导数。将坎尼 3 个准则相结合可以获得最优的检测算子。坎尼边缘检测的算法步骤如下：

（1）用高斯滤波器平滑图像；

（2）用一阶偏导的有限差分来计算梯度的幅值和方向；

（3）对梯度幅值进行非极大值抑制；

（4）用双阈值算法检测和连接边缘。

坎尼算子也可以用 MATLAB 中的 edge 函数来检测边缘：

```
BW = edge(I,'canny',thresh,sigma)
```

式中 thresh 是边缘检测的阈值，sigma 是高斯滤波器的σ值，默认为 2。

【例 4-3】用 Canny 算子检测图像 4-1(a)的边缘，检测结果如图 4-3(c)所示。

```
I = imread('bacteria.BMP');
BW1 = edge(I,'canny',0.2);
figure,imshow(BW1);
```

比较前面梯度算子、LoG 算子和坎尼算子，如图 4-1(b)、4-2 和 4-3，可以发现坎尼算子的边缘检测结果最满意。根据坎尼算子的 3 个准则，在不同的实际应用中，读者也可以提出自己的最优边缘检测算子。

4.3 灰度阈值分割

4.3.1 阈值分割介绍

1. 阈值化分割原理

阈值化分割算法是图像分割中应用数量最多的一类。简单地说，对灰度图像地阈值分割就是先确定一个处于图像灰度取值范围内的灰度阈值，然后将图像中各个像素的灰度值与这个阈值相比较，并根据比较的结果将对应的像素划分（分割）为两类：像素灰度大于阈值的一类，像素灰度值小于阈值的为另一类，灰度值等于阈值的像素可以归入这两类之一。分割后的两类像素一般分属图像的两个不同区域，所以对像素根据阈值分类达到了区域分割的目的。由此可见，阈值分割算法主要有两个步骤：

（1）确定需要分割的阈值；

（2）将分割阈值与像素点的灰度值比较，以分割图像的像素。

以上步骤中，确定阈值是分割的关键，如果能确定一个合适的阈值就可准确地将图像分割开来。阈值确定后，将阈值与像素点的灰度值比较和像素分割可对各像素并行地进行，分割的结果直接给出图像区域。

在利用取阈值方法来分割灰度图像时一般对图像的灰度分布有一定的假设，或者说是基于一定的图像模型。最常用的模型可描述如下：假设图像由具有单峰灰度分布的目标和背景组成，处于目标和背景内部相邻像素间的灰度值是高度相关的，但处于目标和背景交界处两边的像素在灰度值上有很大的差别。若一幅图像满足这些条件，它的灰度直方图基本上可看作是由分别对应于目标和背景的两个单峰直方图混合构成的。而且如果这两个分布大小（数量）接近且均值相距足够远，两部分的均方差也足够小，则直方图应为较明显的双峰。类似地，如果图像中有多个单峰灰度分布的目标，则直方图有可能表现为较明显的多峰。对这类图像常可用取阈值方法来较好地分割。

把图像中各种灰度的像素分成两个不同的类，需要确定一个阈值。如果要把图像中各种灰度的像素分成多个不同的类，那么需要选择一系列阈值以将每个像素分到合适的类别中去。如果只用一个阈值分割称为单阈值分割方法，如果用多个阈值分割称为多阈值分割方法。单阈值分割可看作是多阈值分割的特例，许多单阈值分割算法可推广以进行多阈值分割。反之，有时也可将多阈值分割问题转化为一系列单阈值分割问题来解决。不管用何种方法选取阈值（后续将仔细讨论），一幅原始图像 $f(x,y)$ 取单阈值 T 分割后的图像可定义为：

$$g(x,y)=\begin{cases}1 & f(x,y)>T\\ 0 & f(x,y)\leqslant T\end{cases} \tag{4-16}$$

这样得到的 $g(x,y)$ 是一幅二值图像。

在一般的多阈值分割情况下，取阈值分割后的图像可表示为：

$$g(x,y)=k \qquad T_{k-1}\leqslant f(x,y)<T_k \qquad k=1,2,\cdots,K \tag{4-17}$$

其中 $T_0,T_1,\cdots,T_k$ 是一系列分割阈值，k 表示赋予分割后图像各区域的不同标号。

需要指出，无论是单阈值分割或多阈值分割，分割结果中都有可能出现不同区域具有相同标号或区域值的情况。这是因为取阈值分割时只考虑了像素本身的值，未考虑像素的空间位置。所以根据像素值划分到同一类的像素有可能分属于图像中不相连通的区域。这时往往需要借助一些对场景的先验知识来进一步确定目标区域。

2. 阈值分割算法分类

已提出的阈值化分割算法很多，相应的分类方法也很多，例如，对文档图像的阈值化技术可有 5 种分类的方法：

（1）考虑分割过程是否需要人工干预，可分为交互的与自动的；

（2）根据阈值的不同作用范围，可分为全局的与局部的；

（3）考虑阈值选取中所采用的灰度分布统计特性，可分为上下文相关的与上下文无关的，前者基于灰度分布的一阶统计，后者基于灰度分布的二阶统计；

（4）从处理策略角度考虑，可分为迭代的与非迭代的；

（5）根据为进行分割是否选用训练像素集以估计目标或背景的特性参数，可分为有监督的与无监督的。

在前面关于阈值化原理的讨论中已指出选取合适的分割阈值是阈值化算法的关键问题，据此可根据阈值选取本身的特点对算法分类。阈值一般可写成如下形式：

$$T=T[x,y,f(x,y),p(x,y)] \tag{4-18}$$

其中 $f(x,y)$ 代表在像素点 (x,y) 处的灰度值，$p(x,y)$ 代表在该点邻域的某种局部性质。即阈值 T 在一般情况下可以是 (x,y)，$f(x,y)$ 和 $p(x,y)$ 的函数。借助上式，可将取阈值分割方法分成如下 3 类，相应的阈值分别称为：

（1）基于各像素值的阈值　阈值仅根据 $f(x,y)$ 来选取，所得到的阈值仅与全图各像素的本身性质（像素值）有关；

（2）基于区域性质的阈值　阈值是根据 $f(x,y)$ 和 $p(x,y)$ 来选取的，所得的阈值与区域性质（区域内各像素的值，相邻像素值的关系等）有关；

（3）基于坐标位置的阈值　阈值进一步（除根据 $f(x,y)$ 和 $p(x,y)$ 来选取外）还与 x,y 有关，

即阈值要考虑位置(x,y)来确定，则所得的阈值是与像素空间坐标有关的。

上面确定第一类阈值的技术有时称为点相关技术，而确定第二类阈值的技术有时称为区域相关技术，确定第三类阈值的技术有时称为动态阈值技术。前两类阈值也有称为全局阈值（或称固定阈值）的，因为此时确定的阈值对全图使用，或者说，对各个像素使用相同的阈值来分割；与此对应，第三类阈值也有叫局部阈值或动态阈值的，因为此时确定的阈值是针对图像中的局部区域（也可是单个像素）的。在一定意义上可以认为局部阈值化是全局阈值化技术的局部化特例。

以上对取阈值分割方法的分类思想是通用的。近年来，许多取阈值分割算法借用了视觉特性、神经网络和模糊数学等工具，但仍可把它们归纳到以上 3 种方法类型中。

4.3.2 全局阈值

对灰度图像，基于各像素值的阈值是仅考虑各像素本身灰度值而确定的，因而算法一般较简单，但抗噪声能力不强。所确定的阈值（对多阈值分割是阈值序列）作用于整幅图像的每个像素，因而对目标和背景的灰度有梯度变化的图像效果较差或失效。

图像的灰度直方图是图像各像素灰度值的一种统计度量。许多常用的阈值选取方法就是根据直方图来进行的。如果对双峰直方图选取两峰之间的谷所对应的灰度值作为阈值就可将目标和背景分开（多峰直方图时也类似）。谷的选取有许多方法，下面介绍 3 种比较典型的方法。

1. 极小值点阈值

将图像的灰度直方图的包络看作一条曲线，则选取直方图的谷可借助求曲线极小值的方法。设用 $h(z)$代表直方图，那么极小值点应同时满足：

$$\frac{\partial h(z)}{\partial z}=0 \quad 和 \quad \frac{\partial^2 h(z)}{\partial z^2}>0 \tag{4-19}$$

和这些极小值点对应的灰度值就可用作分割阈值。

实际图像的直方图由于图像噪声等原因经常有很多起伏，使得按式（4-19）计算出的极小值点有可能对应虚假的谷。解决的方法之一是先对直方图进行平滑处理。

2. 最优阈值

有时图像中目标和背景的灰度值有部分交错，这时如用一个全局阈值进行分割则总会产生一定的误差。实际中常希望能尽可能减小误分割（包括把目标分成背景和把背景分成目标两类）的概率，而选取最优阈值是一种常用的方法。这里最优阈值指能使误分割率最小的分割阈值。图像的直方图可看成像素灰度值的概率分布密度函数的一个近似，设一幅图像仅包含两类主要的灰度值区域（目标和背景），那么其直方图所代表的像素灰度值概率分布密度函数实际上是对应目标和背景的两个单峰分布密度函数之和。如果已知密度函数的形式，就有可能计算出一个最优阈值，用它可把图像分成两类区域而使误分割率最小。

设有这样一幅混有加性高斯噪声的图像，背景和目标的概率密度分别是 $p_1(z)$和 $p_2(z)$，整幅图像的混合概率密度：

$$p(z) = P_1 p_1(z) + P_2 p_2(z)$$

$$= \frac{P_1}{\sqrt{2\pi}\sigma_1} \exp\left[-\frac{(z-\mu_1)^2}{2\sigma_1^2}\right] + \frac{P_2}{\sqrt{2\pi}\sigma_2} \exp\left[-\frac{(z-\mu_2)^2}{2\sigma_2^2}\right] \tag{4-20}$$

其中μ_1和μ_2分别是背景和目标区域的平均灰度值，σ_1和σ_2分别是关于均值的均方差，P_1和P_2分别是背景和目标区域灰度值的先验概率。根据概率定义有 $P_1+P_2=1$，所以混合概率密度式（4-20）中只有 5 个未知的参数。如果能求得这些参数就可以确定混合概率密度。

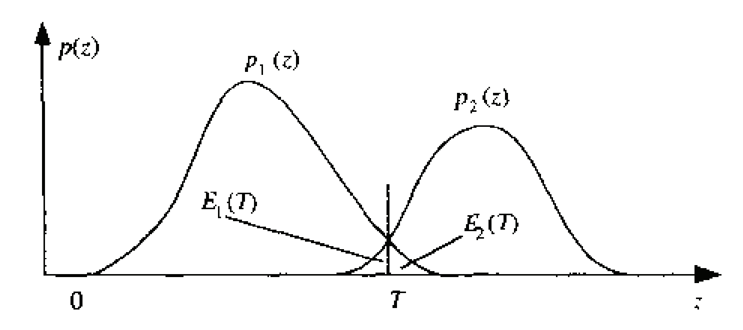


图 4-4　最优阈值选取示意图

如图 4-4 所示。假设$\mu_1<\mu_2$，需确定一个阈值 T 使得灰度值小于 T 的像素分割为背景而使得灰度值大于 T 的像素分割为目标。这时错误地将目标像素划分为背景的概率和将背景像素错误地划分为目标的概率分别是：

$$E_1(T) = \int_{-\infty}^{T} p_2(z)\mathrm{d}z \tag{4-21}$$

$$E_2(T) = \int_{T}^{\infty} p_1(z)\mathrm{d}z \tag{4-22}$$

总的误差概率为：

$$E(T) = P_2 E_1(T) + P_1 E_2(T) \tag{4-23}$$

为求得使该误差最小的阈值可将 $E(T)$对 T 求导并令导数为零，这样得到：

$$P_1 p_1(T) = P_2 p_2(T) \tag{4-24}$$

将这个结果用于高斯密度（即将式（4-20）代入）可解得：

$$\ln\frac{P_1\sigma_2}{P_2\sigma_1} - \frac{(T-\mu_1)^2}{2\sigma_1^2} = -\frac{(T-\mu_2)^2}{2\sigma_1^2} \tag{4-25}$$

当$\sigma_1=\sigma_2=\sigma$ 时：

$$T = \frac{\mu_1+\mu_2}{2} + \frac{\sigma^2}{\mu_2-\mu_1}\ln\frac{P_1}{P_2} \tag{4-26}$$

若先验概率相等，即 $P_1=P_2$，则：

$$T = \frac{\mu_1+\mu_2}{2} \tag{4-27}$$

这表示如果图像灰度值服从正态分布时，最佳阈值可按上式求得。

3. 迭代阈值分割

阈值也可以通过迭代计算得到。首先选取图像的灰度范围的中值作为初始值 T_0，然后按下式迭代：

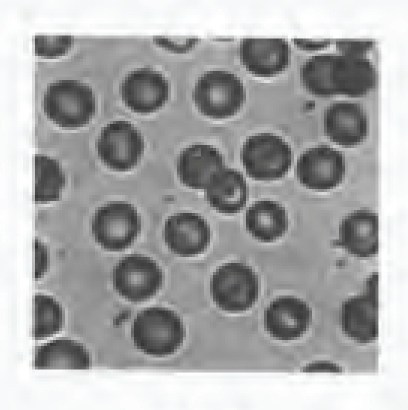
$$T_{i+1}=\frac{1}{2}\left\{\frac{\sum_{k=0}^{T_i}h_k\cdot k}{\sum_{k=0}^{T_i}h_k}+\frac{\sum_{k=T_i+1}^{L-1}h_k\cdot k}{\sum_{k=T_i+1}^{L-1}h_k}\right\} \tag{4-28}$$

式中 h_k 是灰度为 k 值的像素个数，共有 L 个灰度级。迭代一直进行到 $T_{i+1}=T_i$ 结束，取结束时的 T_i 为阈值。本书将在第 7 章中详细介绍迭代阈值的算法和应用实例。

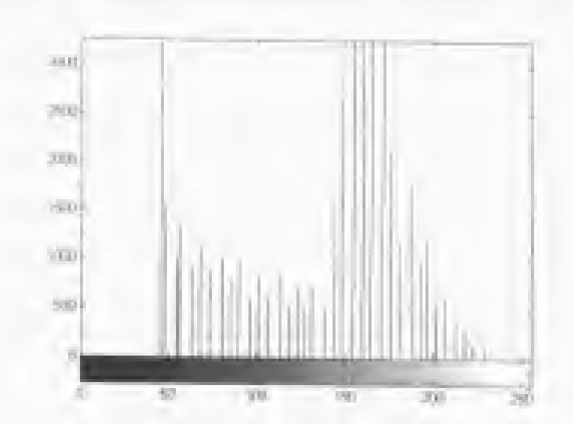
上述方法所确定的阈值作用于整幅图像的每个像素，因而对目标或背景的灰度有剧烈变化的图像，阈值分割效果比较差，甚至失效。

【例 4-4】对图像 4-5(a)进行阈值分割。其灰度直方图如图 4-5(b)所示，阈值分割结果如图 4-5(c)所示。

```
I=imread('blood1.tif');
imhist(I);          % 观察灰度直方图，灰度 140 处有谷，确定阈值 T=140
I1=im2bw(I,140/255); % im2bw 函数需要将灰度值转换到[0,1]范围内
figure,imshow(I1);
```



(a) 原始图像



(b) 原始图像的灰度直方图



(c) 阈值分割结果

图 4-5

4.3.3 动态阈值

当图像中有如下一些情况：有阴影、光照不均匀、各处的对比度不同、突发噪声、背景灰度变化等，如果只用一个固定的全局阈值对整幅图进行分割，则由于不能兼顾图像各处的情况而使分割效果受到影响。有一种解决办法是用与坐标相关的一组阈值（即阈值是坐标的函数）来对图像各部分分别进行分割。这种与坐标相关的阈值也叫动态阈值，这种取阈值分割方法也叫变化阈值法（也有称自适应阈值法的）。这类算法的时间复杂度和空间复杂度比较大，但抗噪声能力强，对一些用全局阈值法不易分割的图像（如目标和背景的灰度有梯度变化的图像）有较好的效果。这种动态阈值化方法在二值化文档图像分割时有较好的性能。

一种比较简单的动态阈值算法是对每个像素确定以它为中心的一个窗口，计算窗口内的最大值和最小值，再取它们的平均值作为该点的阈值，可以证明图像像素点灰度值和该阈值的差具有二阶导数的性质，所以取差的过零点就可得到二值分割结果。下面再介绍另外两个方法，以开阔读者的思路。

1. 阈值插值

可以将变化阈值技术当作全局固定阈值技术的局部化特例。首先将图像分解成一系列子图像，这些子图像可以互相重叠也可以只相接。如果子图像比较小，则由阴影或对比度空间变化等带来的问题就会比较小。然后可对每个子图像计算一个阈值，此时阈值可用任一种固定阈值法选取。通过对这些子图像所得阈值的插值就可得到对图像中每个像素进行分割所需的阈值。这里对应每个像素的阈值合起来组成图像（幅度轴）上的一个曲面，也可叫阈值曲面。一种方法的具体步骤如下：

（1）将整幅图像分成一系列互相之间有 50%重叠的子图像；

（2）做出每个子图像的直方图；

（3）检测各个子图像的直方图是否为双峰，如是可采用上述介绍的的阈值选取方法确定子图像的阈值，否则不进行处理；

（4）根据对直方图为双峰的子图像得到的阈值，通过插值得到所有子图像的阈值；

（5）根据各子图像的阈值，再通过插值得到所有像素的阈值，然后对图像进行分割。

2. 水线阈值算法

水线（也称为分水岭或流域，Watershed）阈值算法和直接在最佳阈值处分割不同，它是一种特殊的自适应迭代阈值分割算法。利用图 4-6 来说明水线阈值算法的原理。

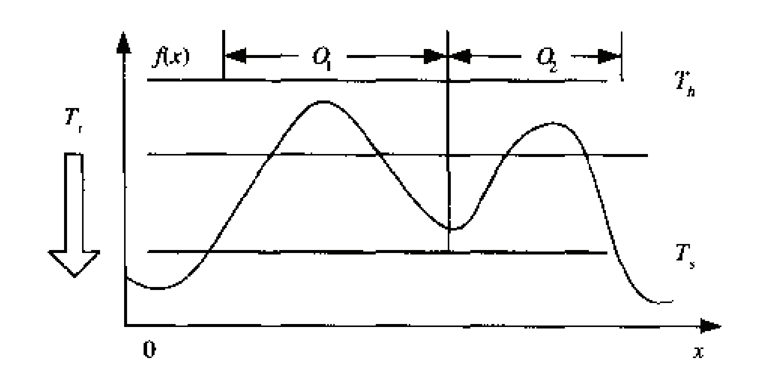


图 4-6　水线阈值算法示意图

图 4-6 中给出的是一幅图像中的一个剖面，其中灰度较高的两个峰分别对应目标 O_1 和 O_2。分割的任务是将两个目标从背景中提取出来并互相分开。先用一个较大的阈值 T_l 进行分割，它可将图中的两个目标与背景分开，只是其间的间隙太宽。如果接下来逐渐减小阈值，目标的边界会随阈值的减少而相向扩展，最终两个目标会接触（相遇），但此时不让两个目标合并，这样它们相接触前所保留的最后像素集合就给出两目标间最终的边界。上述过程在将阈值减小到背景灰度前就可结束。

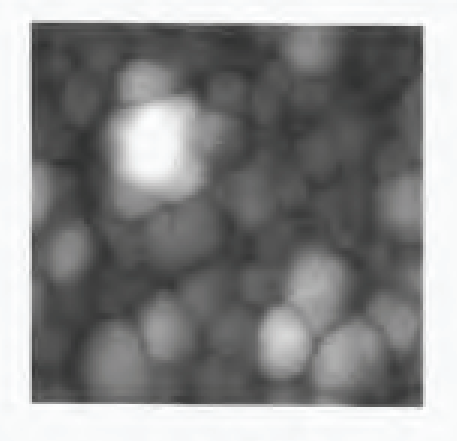
实际中水线算法首先用一个比较高但得到的结构仍能把每个目标孤立开的阈值进行分割。然后，当阈值逐渐减小并逼近最佳阈值时，不再合并原已分开的目标。这样就可解决采用全局阈值方法在目标很接近时造成的目标合并问题。这里初始阈值的选取非常重要，只要初始阈值选合适，那么就可保证最终分割结果的正确性。如果初始阈值选得太大（如图 4-6 中的 T_h），那么低反差的目标在开始时会被漏掉，其后在减少阈值的过程中会被合并；反之，如果初始阈值选得太小（如图 4-6 中 T_s 所示），那么目标在开始时就会被合并。另外最终阈值的选取也很重要，它确定了最终边界与目标吻合的情况。

MATLAB 中用函数 watershed 找到流域，调用格式：L = watershed(A)。

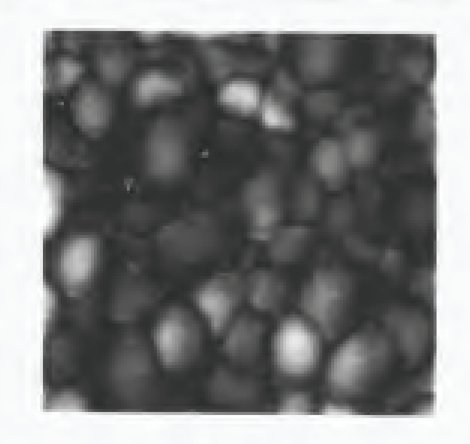
阈值分割的方法很多，每一种方法几乎都有其独特指出和实际应用的背景，此处不再一一介绍。实际应用中，阈值分割经常需要和其他方法相互结合使用，才能取得最佳或满意的分割结果。因此，在介绍完图像处理和识别的基本知识后，有关阈值分割的实际应用在后续的第 7 章、第 8 章和第 9 章中将会结合实际工程项目做详细介绍。下面一个水线阈值法的例子提供参考。

【例 4-5】用水线阈值法分割图像 4-7(a)。图像高帽变换的结果如图 4-7(b)所示，低帽变换的结果如图 4-7(c)所示，高帽变换与低帽变换相减的结果如图 4-7(d)所示，阈值分割相互分离出的目标如图 4-7(e)所示，最后用不同的颜色加以区分如图 4-7(f)所示。

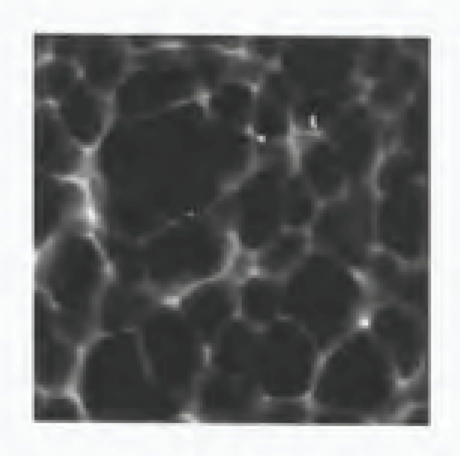
```
afm = imread('afmsurf.tif');figure, imshow(afm);
se = strel('disk', 15);
Itop = imtophat(afm, se);  % 高帽变换
Ibot = imbothat(afm, se);  % 低帽变换
figure, imshow(Itop, []);   % 高帽变换，体现原始图像的灰度峰值
figure, imshow(Ibot, []);   % 低帽变换，体现原始图像的灰度谷值
Ienhance = imsubtract(imadd(Itop, afm), Ibot);% 高帽图像与低帽图像相减，增强图像
figure, imshow(Ienhance);
Iec = imcomplement(Ienhance); % 进一步增强图像
Iemin = imextendedmin(Iec, 20); figure,imshow(Iemin) % 搜索 Iec 中的谷值
Iimpose = imimposemin(Iec, Iemin);
wat = watershed(Iimpose);  % 分水岭分割
rgb = label2rgb(wat); figure, imshow(rgb); %  用不同的颜色表示分割出的不同区域
```



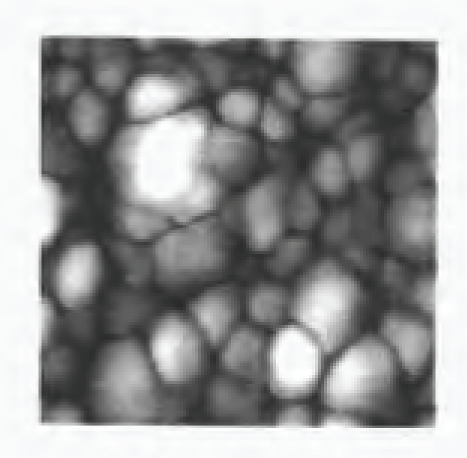
(a) 原始图像



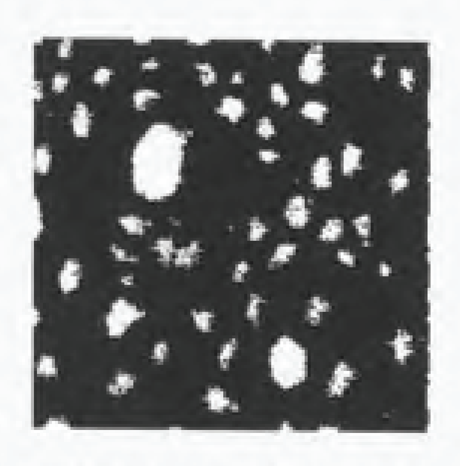
(b) 高帽变换图像



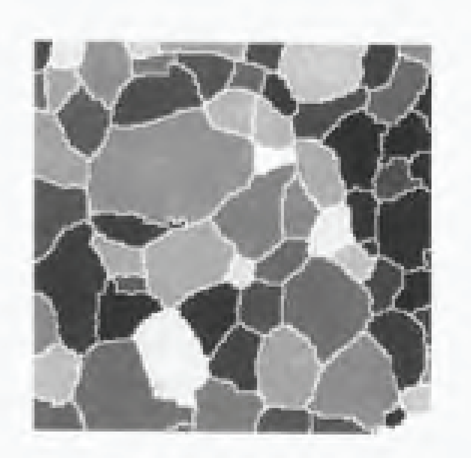
(c) 低帽变换图像



(d) 高帽与低帽相减后图像



(e) 阈值分割出相互分离的目标图像



(f) 水线阈值分割图像

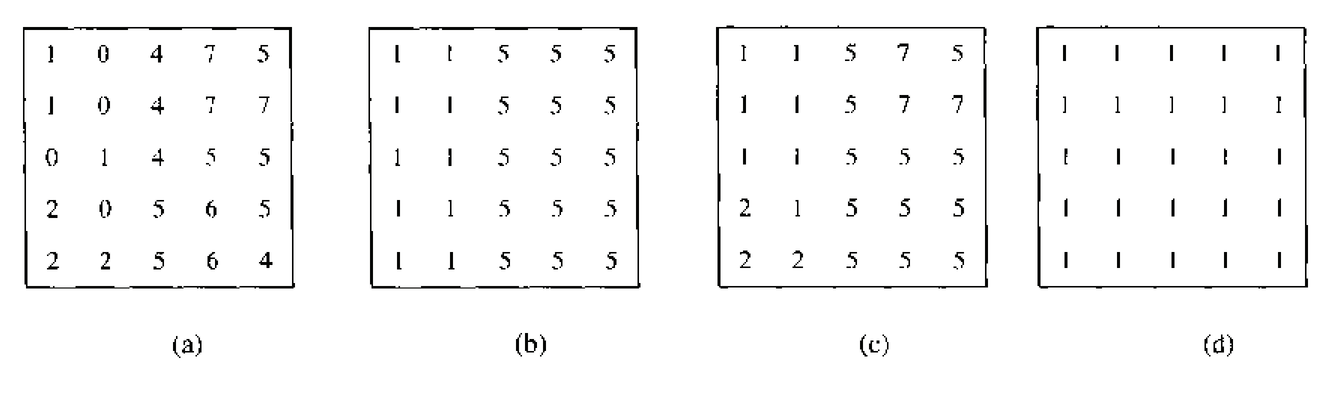
图 4-7

4.4　区域分割

4.4.1　区域生长的原理和步骤

区域生长的基本思想是将具有相似性质的像素集合起来构成区域。具体先对每个需要分割的区域找一个种子像素作为生长的起点，然后将种子像素周围邻域中与种子像素有相同或相似性质的像素（根据某种事先确定的生长或相似准则来判定）合并到种子像素所在的区域中。将这些新像素当作新的种子像素继续进行上面的过程，直到再没有满足条件的像素可被包括进来。这样一个区域就长成了。

图 4-8 给出已知种子点进行区域生长的一个示例。图 4-8(a)给出需分割的图像，设已知有两个种子像素（标为深浅不同的灰色方块），现要进行区域生长。设这里采用的判断准则是：如果所考虑的像素与种子像素灰度值差的绝对值小于某个门限 T，则将该像素包括进种子像素所在区域。图 4-8 (b)给出了 T=3 时的区域生长结果，整幅图被较好地分成 2 个区域；图 4-8(c)给出了 T=1 时的区域生长结果，有些像素无法判定；图 4-8 (d)给出了 T=6 时的区域生长结果，整幅图都被分在一个区域中了。由此例可见门限的选择是很重要的。



(a)

1	0	4	7	5
1	0	4	7	7
0	1	4	5	5
2	0	5	6	5
2	2	5	6	4

(b)

1	1	5	5	5
1	1	5	5	5
1	1	5	5	5
1	1	5	5	5
1	1	5	5	5

(c)

1	1	5	7	5
1	1	5	7	7
1	1	5	5	5
2	1	5	5	5
2	2	5	5	5

(d)

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

图 4-8　区域生长示例

从上面的示例可知，在实际应用区域生长方法时需要解决 3 个问题：①如何选择一组能正确代表所需区域的种子像素；②如何确定在生长过程中能将相邻像素包括进来的准则；③如何确定生长终止的条件或规则。

第一个问题通常可以根据具体图像的特点来选取种子像素。例如，在红外图像检测技术中，通常目标的辐射都比较大，所以可以选择图像中最亮的像素作为种子像素。如果没有图像的先验知识，那么可以借助生长准则对像素进行相应的计算。如果计算结果可以看出聚类的情况，那么可以选择聚类中心作为种子像素。

第二个问题的解决不但依赖于具体问题的特征，还与图像的数据类型有关。如果图像是 RGB 彩色图像，那么如果使用单色准则就会影响分割结果。另外，还需要考虑像素间的连通性否则有时会出现无意义的分割结果。后续 4.4.2 小节将介绍几种典型的生长准则和对应的生长过程。

一般生长过程在进行到再没有满足生长准则需要的像素时停止。但常用的基于灰度、纹理、彩色的准则大都基于图像中的局部性质，并没有充分考虑生长的“历史”。为增加区域生长的能力常需考虑一些与尺寸、形状等图像和目标的全局性质有关的准则，在这种情况下常需对分割结果建立一定的模型或辅以一定的先验知识。

4.4.2 生长准则和过程

区域生长的一个关键是选择合适的生长或相似准则，大部分区域生长准则使用图像的局部性质。生长准则可根据不同原则制定，而使用不同的生长准则会影响区域生长的过程。下面介绍 3 种基本的生长准则和方法。

1. 灰度差准则

区域生长方法将图像以像素为基本单位来进行操作，基于区域灰度差的方法主要有如下步骤：

（1）对图像进行逐行扫描，找出尚没有归属的像素；

（2）以该像素为中心检查它的邻域像素，即将邻域中的像素逐个与它比较，如果灰度差小于预先确定的阈值，将它们合并；

（3）以新合并的像素为中心，返回到步骤（2），检查新像素的邻域，直到区域不能进一步扩张；

（4）返回到步骤（1），继续扫描直到不能发现没有归属的像素，则结束整个生长过程。

采用上述方法得到的结果对区域生长起点的选择有较大依赖性。为克服这个问题可采用下面的改进方法：

（1）设灰度差的阈值为零，用上述方法进行区域扩张，使灰度相同像素合并；

（2）求出所有邻接区域之间的平均灰度差，并合并具有最小灰度差的邻接区域；

（3）设定终止准则，通过反复进行上述步骤②中的操作将区域依次合并直到终止准则满足为止。

另外，当图像中存在缓慢变化的区域时，上述方法有可能会将不同区域逐步合并而产生错误。为克服这个问题，可不用新像素的灰度值去与邻域像素的灰度值比较，而用新像素所在区域的平均灰度值去与各邻域像素的灰度值进行比较。

对一个含 N 个像素的图像区域 R，其均值为：

$$m=\frac{1}{N}\sum_{R}f(x,y) \tag{4-29}$$

对像素是否合并的比较测试表示为：

$$\max_{R}\left|f(x,y)-m\right|<T \tag{4-30}$$

其中 T 为给定的阈值。

区域生长的过程中，要求图像的同一区域的灰度值变化尽可能小，而不同的区域之间，灰度差尽可能大。两种情况进行讨论：

（1）设区域为均匀的，各像素灰度值为均值 m 与一个零均值高斯噪声的叠加。当用式（4-30）测试某个像素时，条件不成立的概率为：

$$P(T)=\frac{2}{\sqrt{2\pi}\sigma}\int_{T}^{\infty}\exp(-\frac{z^{2}}{2\sigma^{2}})dz \tag{4-31}$$

这就是误差概率函数，当 T 取 3 倍的方差时，误判概率为 1～99.7%。这表明，当考虑灰度均值时，区域内的灰度变化应尽量小。

（2）设区域为非均匀，且由两部分不同目标的图像像素构成。这两部分像素在 R 中所占比例分别为 q_1 和 q_2，灰度值分别为 m_1 和 m_2，则区域均值为 $q_1m_1+q_2m_2$。对灰度值为 m 的像素，它与区域均值的差为：

$$S_m = m_1 - (q_1 m_1 + q_2 m_2) \tag{4-32}$$

根据式（4-30），可知正确的判决概率为：

$$P(T) = \frac{1}{2}\left[P(|T - S_m|) + P(|T + S_m|)\right] \tag{4-33}$$

这表明，当考虑灰度均值时，不同部分像素间的灰度差距应尽量大。

2. 灰度分布统计准则

这里考虑以灰度分布相似性作为生长准则来决定区域的合并，具体步骤为：

（1）把图像分成互不重叠的小区域；

（2）比较邻接区域的累积灰度直方图根据灰度分布的相似性进行区域合并；

（3）设定终止准则，通过反复进行步骤（2）中的操作将各个区域依次合并直到终止准则满足。

这里对灰度分布的相似性常用两种方法检测（设 $h_1(z)$，$h_2(z)$分别为两邻接区域的累积灰度直方图）：

Kolmogorov—Smirnov 检测

$$\max_z |h_1(z) - h_2(z)| \tag{4-34}$$

Smoothed-Difference 检测

$$\sum_z |h_1(z) - h_2(z)| \tag{4-35}$$

如果检测结果小于给定的阈值，即将两区域合并。

采用灰度分布相似判别准则合并法形成区域的处理过程与灰度差判别准则的合并法相类似。灰度分布相似合并法生成区域的效果与微区域的大小和阈值的选取关系密切，一般说来，微区域太大，会造成因过渡合并而漏分区域；反之，则因合并不足而割断区域。而且，图像的复杂程度，原图像生成状况的不同，对上述参数的选择会有很大影响。通常，微区域大小 q 和阈值 T 由特定条件下的区域生成效果确定。

3. 区域形状准则

在决定对区域的合并时也可以利用对目标形状的检测结果，常用的方法有两种：

（1）把图像分割成灰度固定的区域，设两邻接区域的周长分别为 P_1 和 P_2，把两区域共同边界线两侧灰度差小于给定值的那部分长度设为 L，如果（T_1 为预定阈值）：

$$\frac{L}{\min(P_1, P_2)} > T_1 \tag{4-36}$$

则合并两区域；

（2）把图像分割成灰度固定的区域，设两邻接区域的共同边界长度为 B，把两区域共同

边界线两侧灰度差小于给定值的那部分长度设为 L，如果（T_2 为预定阈值）

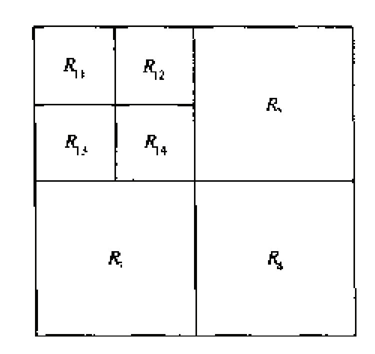
$$\frac{L}{B} > T_2 \tag{4-37}$$

则合并两区域。

上述两种方法的区别是：第一种方法是合并两邻接区域的共同边界中对比度较低部分占整个区域边界份额较大的区域，而第二种方法则是合并两邻接区域的共同边界中对比度较低部分比较多的区域。

4.4.3 分裂合并

4.4.2 节介绍的区域生长方法是先从单个种子像素开始通过不断接纳新像素最后得到整个区域。另一种分割的想法可以是先从整幅图像开始通过不断分裂得到各个区域。实际中常先把图像分成任意大小且不重叠的区域，然后再合并或分裂这些区域以满足分割的要求。



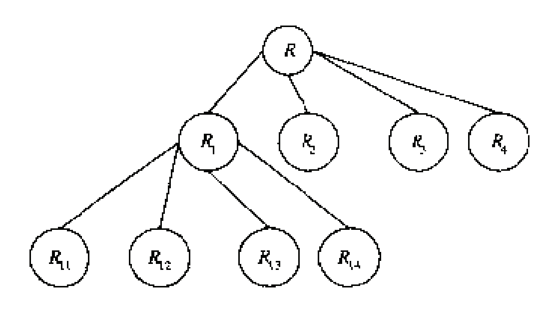


图 4-9　图像的四叉树分解示意图

在这类方法中，最常用的方法四叉树分解法。设 R 代表整个正方形图像区域（如图 4-9 所示），P 代表逻辑谓词。从最高层开始，把 R 连续地分裂成越来越小的 1/4 的正方形子区域 R_i，并且始终使 $P(R_i)$=TRUE。换句话说，如果 $P(R_i)$=FALSE，那么就将图像分成 4 等分。如果 $P(R_i)$=FALSE，那么就将 R 分成 4 等分。如此类推，直到 R_i 为单个像素。

如果仅仅允许使用分裂，最后有可能出现相邻的两个区域具有相同的性质但并没有合成一体的情况。为解决这个问题，在每次分裂后允许其后继续分裂或合并。这里合并只合并那些相邻且合并后组成的新区域满足逻辑谓词 P 的区域。换句话说，如果能满足条件 $P(R_i \cup R_j)$=TRUE，则将 R_i 和 R_j 合并起来。

总结前面所述的基本分裂合并算法步骤如下：

（1）对任一个区域 R_i，如果 $P(R_i)$=FALSE 就将其分裂成不重叠的 4 等分；

（2）对相邻的两个区域 R_i 和 R_j 它们也可以大小不同，即不在同一层），如果条件 $P(R_i \cup R_j)$=TRUE 满足，就将它们合并起来；

（3）如果进一步的分裂或合并都不可能了，则结束。

MATLAB 中可以调用图像处理工具中的 qtdecomp 函数来实现四叉树分解。这个函数首先将图像划分为相等大小的正方形，然后对每一个块进行测试，观察它们是否与标准具有相同性。对不符合标准的块进行进一步分割，重复执行至每一个块都符合标准为止。

qtdecomp 的基本调用方法如下：

```
S=qtdecomp(I, Threshold, [MinDim MaxDim])
```

其中 I 是输入图像。Threshold 是一个可选参数，如果某个子区域中的最大的像素灰度值减去最小的像素灰度值大于 Threshold 设定的阈值，那么继续进行分解，否则停止并返回。[MinDim MaxDim]也是可选参数，用来指定最终分解得到的子区域大小，返回值 S 是一个稀疏矩阵，其非零元素的位置回应于块的左上角，每一个非零元素值代表块的大小。

【例 4-6】对下列矩阵进行四叉树分解。

```
I = [ 1    1    1    1    2    3    6    6
      1    1    2    1    4    5    6    8
      1    1    1    1   10   15    7    7
      1    1    1    1   20   25    7    7
     20   22   20   22    1    2    3    4
     20   22   22   20    5    6    7    8
     20   22   20   20    9   10   11   12
     22   22   20   20   13   14   15   16];
S = qtdecomp(I,5);
full(S)
```

分解返回矩阵如下：

```
ans =
     4    0    0    0    2    0    2    0
     0    0    0    0    0    0    0    0
     0    0    0    0    1    1    2    0
     0    0    0    0    1    1    0    0
     4    0    0    0    2    0    2    0
     0    0    0    0    0    0    0    0
     0    0    0    0    2    0    2    0
     0    0    0    0    0    0    0    0
```

4.5　彩色分割

对颜色的感受是人类对电磁辐射中可见部分里不同频率知觉的体现。随着技术的进步，彩色图像使用得越来越多，彩色图像的分割在最近几年也越来越引起人们的重视。许多原用于灰度图像分割的方法并不适合于直接分割彩色图像。现已提出的彩色图像分割方法主要包括聚类法、熵阈值与博弈论标记结合法，区域分裂合并、区域生长、松弛以及边缘检测等。将基于种子点的区域生长法和基于主动轮廓模型的方法结合也可分割彩色图像。利用神经网络的方法也常有报道。

在许多实际应用中，可对彩色图像的各个分量进行适当的组合转化为灰度图像，然后可用对灰度图像的分割算法进行分割。下面仅考虑专门用于彩色图像分割的方法。

要分割一幅彩色图像，首先要选好合适的彩色空间，其次要采用适合于此空间的分割策略和方法。下面分别讨论这两个问题。

4.5.1 分割所用的彩色空间

表达颜色的彩色空间有许多种，它们常是根据不同的应用目的而提出的。下面围绕图像分割，介绍几种常用的彩色空间和它们的特点。

最常用的彩色空间是红绿蓝（RGB）空间，它是一种矩形直角空间结构的模型，是通过对颜色进行加运算完成颜色综合的彩色系统。它用R、G和B 3个基本分量的值来表示颜色，原点对应于黑色，离原点最远的顶点对应于白色（如图4-10所示）。

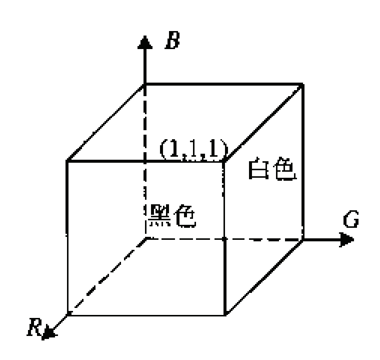


图4-10 RGB彩色立方体

根据这个模型，每幅彩色图包括3个独立的基色平面，或者说分解到3个平面上。与RGB空间对应的是深蓝（Cyan），品红（Mengenta）和黄（Yellow）的CMY彩色空间。RGB和CMY之间的关系转换如下：

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4-38}$$

通过对不同类型的图像分析，有人经过大量实验提出，可用由R、G和B经过线性变换得到的3个正交彩色特征来进行分割。

$$I_1 = \frac{R+G+B}{3}$$

$$I_2 = \frac{R-B}{2} \text{或} I_2 = \frac{B-R}{2} \tag{4-39}$$

$$I_3 = \frac{2G-R-B}{4}$$

这3个特征中，I_1是最佳特征，I_2是次佳特征，仅用I_1和I_2作特征对大多数图像已可得到较好的分割结果。

彩色图像常用R、G和B 3分量的值来表示。但R、G和B 3分量之间常有密切的相关性，直接利用这些分量常常不能得到所需的效果。为了降低彩色特征空间中各个特征分量之间的相关性，以及为了使所选的特征空间更方便于彩色图像分割方法的具体应用，实际中常需要将

RGB 图像变换到其他的彩色特征空间中去。

比较接近人对颜色视觉感知的是色度、饱和度和亮度（Hue、Saturation 和 Intensity，HSI）空间。其中 I 表示颜色的明暗程度（也有用 V(value)表示的），主要受光源脆弱影响，H 表示不同颜色，如黄、红和绿，而 S 表示颜色的深浅如深红、浅红。注意 HSI 模型有两个重要的事实作为基础，首先，I 分量与彩色信息无关，其次 H 和 S 分量与人感受彩色的方式紧密相连。HSI 空间比较直观并且符合人的视觉特性，这些特点使得 HSI 模型非常适合基于人的视觉系统对彩色感知特性的图像处理，HIS 彩色空间表示如图 4-11 所示。从 RGB 到 HSI 的转换关系为：

$$I=\frac{R+G+B}{3}$$

$$S=1-\frac{3}{R+G+B}[\min(R,G,B)] \tag{4-40}$$

$$H=\cos^{-1}\left[\frac{(R-G)+(R-B)}{2\sqrt{(R-G)^2+(R-B)(G-B)}}\right] \quad R\neq B \quad or \quad G\neq B$$

其中 S 也可以表示为：

$$S=\max(R,G,B)-\min(R,G,B) \tag{4-41}$$

式（4-40）中，H 是 R、G 和 B 经非线性变换而得到的。在饱和度低的区域，H 值量化粗，特别是在饱和度为 0 的区域（黑白区域），H 值已没有意义，即当 S=0 时，对应灰度无色，这时 H 没有意义，此时定义 H 为 0。最后当 I=0 时，S 也没有意义。



图 4-11　HIS 颜色空间示意图

在HSI空间中，H、S和I 3分量之间的相关性比R、G和B 3分量之间要小得多。由于HSI彩色空间的表示比较接近人眼的视觉生理特性，人眼对H、S、I变化的区分能力要比对R、G、B变化的区分能力强。另外在HSI空间中彩色图像的每一个均匀性彩色区域都对应一个相对一致的色调（H），这说明色调能够被用来进行独立于阴影的彩色区域的分割。

最后，孟塞尔色空间也用一个三维空间的模型将各种表面色的3种视觉特性：亮度、色度和饱和度全部表示出来。孟塞尔颜色系统的颜色样品在视觉上是均匀的，因而可以用它来考察和验证与某一色差公式有关的颜色空间的均匀性。孟塞尔的色度值、亮度值和彩度值大致反映了物体颜色的心理规律，代表了颜色的色度、亮度和饱和度的主观特性。孟塞尔颜色空间是从心理学角度，根据颜色的视觉特点所制定的颜色分类和标定系统，比较符合人的视觉特性，但孟塞尔空间完全以目视色表为基础，没有数学表达式，使用起来很不方便。

4.5.2 分割策略

测量空间聚类法是分割彩色图像常用的方法。彩色图像在各个空间均可看作由3个分量构成，所以分割彩色图像的一种方法是建立一个“3-D直方图”，它可用一个3-D数组表示。这个3-D数组中的每个元素代表图像中给定3个分量值的像素的个数。阈值分割的概念可以扩展为在3-D空间搜索像素的聚类，并根据聚类来分割图像。

将一幅图像分割成K个区域的一种常用方法是C－均值聚类法。令$x=(x_1,x_2)$代表一个像素的坐标，$g(x)$代表这个像素的灰度值，C－均值法的聚类准则为：

$$E=\sum_{i=1}^{K}\sum_{x\in Q_j^{(i)}}\left\|g(x)-\mu_j^{(i+1)}\right\| \tag{4-42}$$

其中$Q_j^{(i)}$代表在第I次迭代后赋予给类j的像素集合，$\mu_j^{(i+1)}$表示第j类的均值。式（4-42）的指标给出了每个像素与其他对应类均值的距离和。具体的C－均值聚类算法步骤如下：

（1）选取K个聚类中心：$\mu_1^{(1)},\mu_2^{(1)},\cdots,\mu_K^{(1)}$（上角标记为聚类中的迭代次数）。

（2）进行到第i次迭代时，根据下述准则将每个像素都分到K类之一（j=1,2,⋯,K，l=1,2,⋯,K，$l\neq j$），即：

$$x\in Q_l^{(i)}\qquad \text{如果}\left\|g(x)-\mu_l^{(i)}\right\|<\left\|g(x)-\mu_j^{(i)}\right\| \tag{4-43}$$

即将每个像素归到离它最近的类。

（3）计算各聚类中心的新值：

$$\mu_j^{(i+1)}=\frac{1}{N_j}\sum_{x\in Q_j^{(i)}}g(x) \tag{4-44}$$

其中N_j是$Q_j^{(i)}$中的像素个数。

（4）如果对于所有的j=1,2,⋯,K，有$\mu_j^{(i+1)}=\mu_j^{(i)}$，则算法收敛，结束；否则重复执行步骤（2），继续下一次迭代。

C－均值聚类算法的优点是它能够动态聚类，具有一定的自适应性。但是C－均值聚类的结果易受聚中心的个数 *K* 及初始聚类中心的影响，同时也受样本的几何形状及排列次序的影响。因此，算法能否收敛决于样本的特性和其能够形成不同区域的个数。

彩色图像可以分步分割。有一种分两步的彩色图像分割算法，第一步借助取阈值方法进行粗略分割将图像转化为若干个区域，第二步利用模糊C－均值法将第一步剩下的像素进一步分类。这种方法可看作是由粗到细进行的，先用取阈值方法是为了减少运用模糊C－均值聚类所需的计算量。

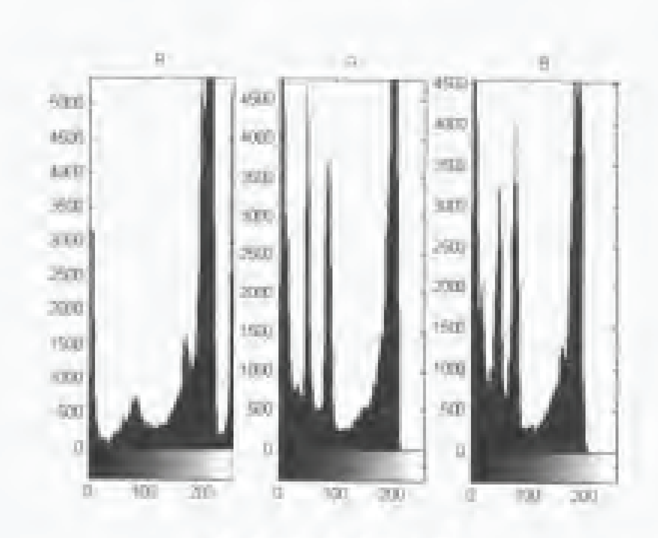
测量空间聚类的方法有一系列优点。首先，将图像由图像空间转换到测量空间的变换常是多对一的变换，这样变换后数据量减少易于计算。其次，尽管许多聚类方法本质上是递归或迭代的，大部分聚类方法可以产生比较光滑的区域边界且比较不受噪声和局部边缘变化的影响。

当对彩色图像的分割在HSI空间进行时，由于H、S和I 3个分量是相互独立的，所以有可能将这个3-D搜索问题转化为3个1-D搜索。

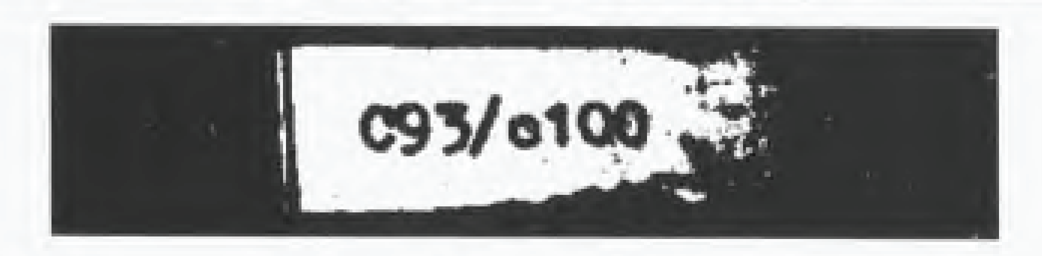
【例4-7】将4-12(a)图像分为文字和非文字的两个区域。其RGB灰度直方图如图4-12(b)所示，分割结果如图4-11(c)所示。



(a) 原始图像



(b) RGB灰度直方图



(c) 聚类分割后的图像

图 4-12

本例中，先通过观察灰度直方图，确定文字和非文字的聚类中心，然后利用欧式距离进行判断聚类。由于图像分割目标简单，不需要进行动态的C－均值聚类。

```
I=imread('4-11.jpg');
I1=I(:,:,1);
I2=I(:,:,2);
I3=I(:,:,3);
[y,x,z]=size(I);

d1=zeros(y,x);
d2=d1;
myI=double(I);
I0=zeros(y,x);
for i=1:x
   for j=1:y
%欧式聚类
d1(j,i)=sqrt((myI(j,i,1)-180)^2+(myI(j,i,2)-180)^2+(myI(j,i,3)-180)^2);
d2(j,i)=sqrt((myI(j,i,1)-200)^2+(myI(j,i,2)-200)^2+(myI(j,i,3)-200)^2);

       if (d1(j,i)>=d2(j,i))
           I0(j,i)=1;
       end
   end
end

figure(1);
imshow(I);
% 显示RGB空间的灰度直方图，确定两个聚类中心(180,180,180)和(200,200,200)
figure(2);
subplot(1,3,1);
imhist(I1);
subplot(1,3,2);
imhist(I2);
subplot(1,3,3);
imhist(I3);
```

```
figure(4);
imshow(I0);
```

上述图像可以先将彩色图像的256×256×256位的颜色空间压缩到16位等较小的空间，然后采用动态C－均值聚类分割，能够得到更好的效果。或者先采用阈值分割，将图像粗略分割后再利用聚类进行分割。

4.6　特殊方法的图像分割

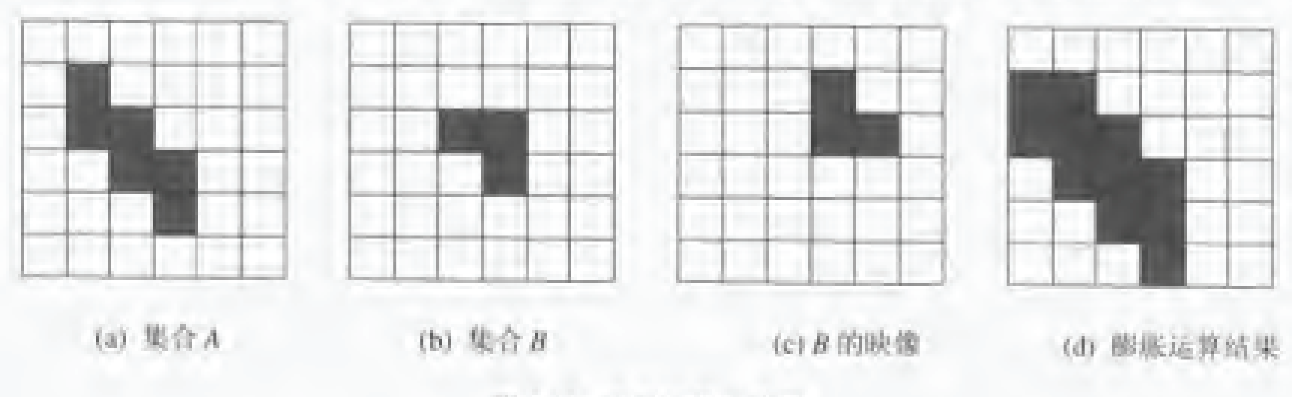
4.6.1　基于数学形态学的分割技术

形态学一般指生物学中研究动物和植物结构的一个分支。后来人们用数学形态学（也称图像代数）表示以形态为基础对图像进行分析的数学工具。它的基本思想是用具有一定形态的结构元素去量度和提取图像中的对应形状以达到对图像分析和识别的目的。数学形态学的数学基础和所用语言是集合论。数学形态学的应用可以简化图像数据，保持它们基本的形状特性，并除去不相干的结构。数学形态学的算法具有天然的并行实现的结构。

数学形态学的基本运算有4个：膨胀（或扩张）、腐蚀（或侵蚀）、开启和闭合。基于这些基本运算还可推导和组合成各种数学形态学实用算法。二值形态学中的运算对象是集合，通常给出一个图像集合和一个结构元素集合，利用结构对图像进行操作。但要注意，实际运算中所使用的两个集合不能看成是互相对等的：如果A是图像，B是结构元素，形态学运算将使用B对A进行操作。结构元素是用来定义形态学中所用到的邻域的形状和大小的矩阵。对某些强噪声图像，基于数学形态学的算法有可能取得较好的效果。

1. 数学形态学的基本运算

数学形态学的基本运算有4个：膨胀（或扩张）、腐蚀（或侵蚀）、开启和闭合。



(a) 集合A　　(b) 集合B　　(c) B的映像　　(d) 膨胀运算结果

图4-13　膨胀运算示意图

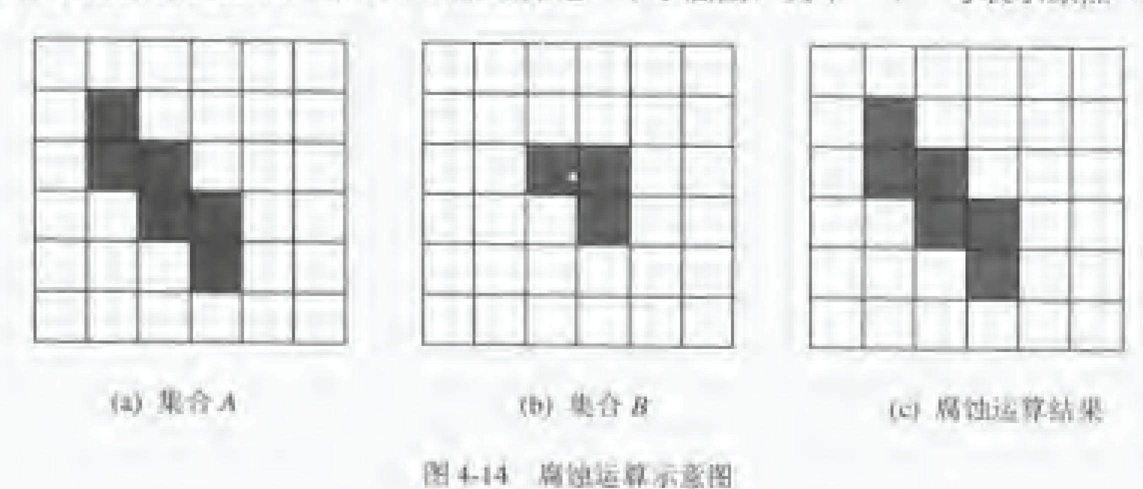
膨胀的运算符为⊕，图像集合A（如图4-13(a)），用结构元素B（如图4-13(b)）来膨胀，记作$A \oplus B$，其定义为：

$$A \oplus B = \left\{ x \mid [(\hat{B})_x \cap A] \neq \varnothing \right\} \tag{4-45}$$

其中，$\hat{B}$ 表示 B 的映像，如图 4-13(c)所示，即与 B 关于原点对称的集合。上式表明，用 B 对 A 进行膨胀的运算过程如下：首先作 B 关于原点的映射，再将其映像平移 x，当 A 与 B 映像的交集不为空时，B 的原点就是膨胀集合的像素。也就是说，用 B 来膨胀 A 得到的集合时集合 $\hat{B}$ 的位移与 A 至少有一个非零元素相交时 B 的原点的位置集合，膨胀运算结果如图 4-13(d)所示。因而（4-45）也可以表示为：

$$A \oplus B = \left\{ x \mid [(\hat{B})_x \cap A] \subseteq A \right\} \tag{4-46}$$

如果将 B 看成是一个卷积模板，膨胀就是对 B 做关于原点的映像，然后再将映像连续地在 A 上移动而实现。图 4-14 给出了膨胀运算地一个示意图，其中“+”号表示原点（下同）。



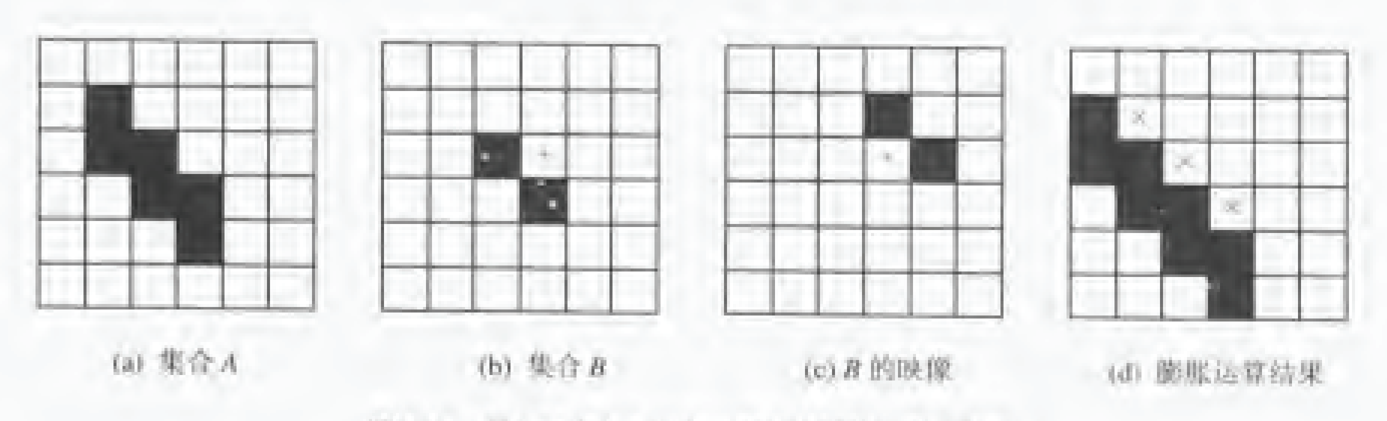
(a) 集合 A　　(b) 集合 B　　(c) 腐蚀运算结果

图 4-14　腐蚀运算示意图

腐蚀的运算符是 Θ，A 用 B 来腐蚀记作 $A\Theta B$，其定义为：

$$A\Theta B = \left\{ x \mid (B)_x \subseteq A \right\} \tag{4-47}$$

上式表明，A 用 B 腐蚀的结果是所有满足将 B 平移后，B 仍旧全部包含在 A 中的 x 的集合，从直观上看就是 B 经过平移后全部包含在 A 中的原点组成的集合。如图 4-15 所示为一个腐蚀运算的示意图。



(a) 集合 A　　(b) 集合 B　　(c) B 的映像　　(d) 膨胀运算结果

图 4-15　原点不包含在结构元素中的膨胀运算示意图

以上讨论中都假设原点是位于结构元素之中的，对于膨胀运算来说，总有：

$$A \subseteq A \oplus B \tag{4-48}$$

而对于腐蚀运算来说，总有：

$$A \Theta B \subseteq A \tag{4-49}$$

如果原点不在结构元素中，运算结果将会发生变化。对于膨胀来说，总有：

$$A \not\subset A \oplus B \tag{4-50}$$

如图 4-14 所示，原点不包含在结构元素中时的膨胀运算示意图。

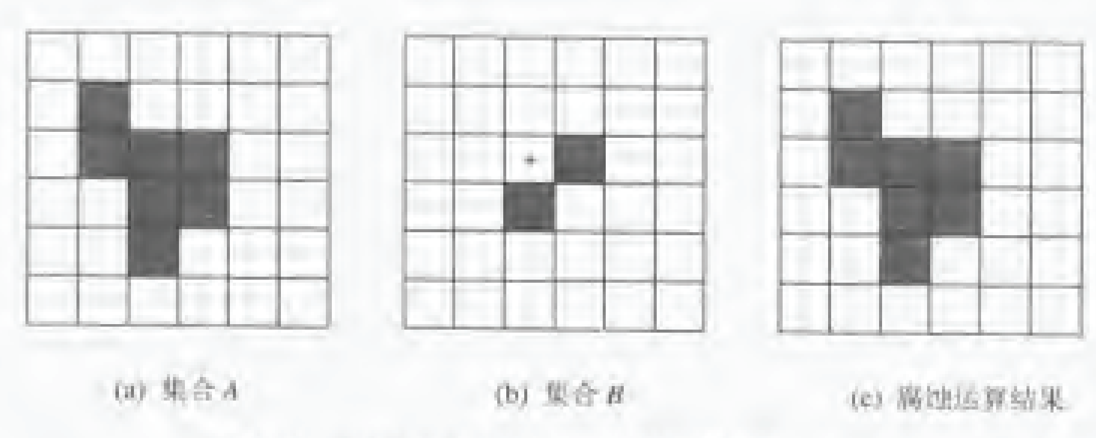
而对于腐蚀运算来说，如果原点不包含在结构元素中，那么会有两种可能，一种是：

$$A \Theta B \subseteq A \tag{4-51}$$

另一种是：

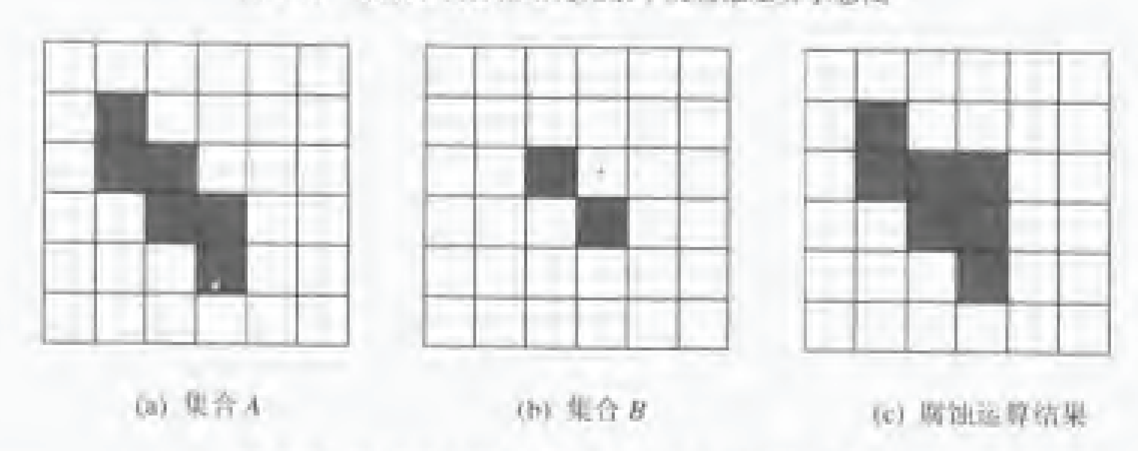
$$A \Theta B \not\subset A \tag{4-52}$$

如图 4-16 和 4-17 分别给出了上述两种情况的腐蚀运算结果示意图。



(a) 集合 A　　(b) 集合 B　　(c) 腐蚀运算结果

图 4-16　原点不包含在结构元素中的腐蚀运算示意图一



(a) 集合 A　　(b) 集合 B　　(c) 腐蚀运算结果

图 4-17　原点不包含在结构元素中的腐蚀运算示意图二

膨胀和腐蚀并不是互为逆运算，它们可以级连结合使用。使用同一个结构元素对图像先进行腐蚀运算然后再进行膨胀的运算称为开启，先进行膨胀然后腐蚀的运算称为闭合。它们也是形态学中的重要运算。

开启的运算符为“$\circ$”，A 用 B 来开启记为 $A \circ B$，其定义如下：

$$A \circ B = (A \Theta B) \oplus B \tag{4-53}$$

闭合的运算符为“•”，A 用 B 来闭合记为 $A \bullet B$，其定义如下：

$$A \bullet B = (A \oplus B)\Theta \qquad (4\text{-}54)$$

开启和闭合不受原点位置的影响，无论原点是否包含在结构元素中，开启和闭合的结果都是一定的。

2. 形态学边缘检测

许多常用的边缘检测算子通过计算图像中局部小区域的差分来工作。这类边缘检测器或算子对噪声都比较敏感并且常常会在检测边缘的同时加强噪声。而形态边缘检测器主要用到形态梯度的概念，虽也对噪声较敏感但不会加强或放大噪声。设用 A 表示图像，B 表示结构元素（A 和 B 均为集合），最基本的形态梯度可定义如下：

$$Grad_1 = (A \oplus B) - (A \Theta B) \qquad (4\text{-}55)$$

较尖锐（细）的边界可用如下两个等价定义的形态梯度获得：

$$Grad_2 = (A \oplus B) - A \qquad (4\text{-}56)$$

$$Grad_3 = A - (A \Theta B) \qquad (4\text{-}57)$$

$Grad_1$、$Grad_2$ 和 $Grad_3$ 都没有放大噪声，但本身仍可能包含不少噪声。下面给出另一种形态梯度：

$$Grad_4 = \min\{[(A \oplus B - A)], [A - (A \Theta B)]\} \qquad (4\text{-}58)$$

这种梯度对孤立的噪声点不敏感，如果将它用于理想斜面边缘检测的检测效果很好。它的缺点是检测不出理想的阶梯边缘，这时可以先对图像进行模糊，将理想阶梯边缘转化为理想斜面边缘后再用 $Grad_4$。

注意：模糊模板的范围（尺寸）与作用于膨胀和腐蚀的结构元素模板保持一致。

MATLAB 中数学形态学的 4 个基本运算：膨胀、腐蚀、开启和闭合，分别可以用 imdilate、imerode、imopen 和 imclose 函数来实现，调用格式如下：

```
J=imdilate(I,SE);
J=imerode(I,SE);
J=imopen(I,SE);
J=imclose(I,SE);
```

其中，SE 表示结构元素。

【例 4-8】用形态学梯度（4-56）式检测二值图像 4-18(a)的边缘。由于腐蚀运算对二值图像种灰度为 1 的像素点进行，先对原始图像取反，如图 4-18(b)所示。然后运用 3×3 结构的腐蚀运算消除边缘，与腐蚀前的图像相减得边缘图像，如图 4-18(c)所示。



(a) 原始图像



(b) 图像取反



(c) 图像边缘检测

图 4-18

```
I=imread('wrod213.bmp');
imshow(I);
I=~I;         % 腐蚀运算对灰度值为 1 的进行
figure, imshow(I);
SE=strel('square',3); % 定义 3×3 腐蚀结构元素
J=imerode(~I,SE);
BW=(~I)-J;         % 检测边缘
figure,imshow(BW);
```

3×3 结构元素对图像的腐蚀相当于去除图像的轮廓或边缘，原始图像与腐蚀后的图像相减，则得到原始图像的边缘。

3. 形状的图像分割

对于图像中形状差别比较大的图像，如矩形、正方形和线条等，可以采用数学形态学进行形状滤波，分割出目标图像。

【例 4-9】从图像 4-19(a)中删除所有电流线，仅保留芯片对象，形态学运算结果如图 4-19(b)所示。



(a)　集成电路原始图像



(b) 形态学分割出的芯片图像

图 4-19

```
I=imread('circbw.tif');
imshow(I);
SE=strel('rectangle',[40 30]);  % 结构定义
J=imopen(I,SE);             % 开启运算
figure,imshow(J);
```

4. 水线区域分割

前面已经介绍水线（分水岭，Watershed）阈值分割方法，下面介绍水线区域分割法，这是一种分割图像中相接触目标的形态学方法（与水线阈值分割法不同），它的基本过程是连续腐蚀二值图像。这个算法包括 3 个步骤。

（1）产生距离图

距离图是图中各个像素的灰度与该像素到图像或目标边界成比例的图。考虑一幅包含目标和背景的二值图，如将较大的值赋予接近目标内部的像素（与距离成正比）就可得到一幅距离图。为用形态学方法产生距离图，可迭代地腐蚀二值图，在每次腐蚀后将所有剩下的像素值加 1。

图 4-20 给出距离图计算的一个示例，如图图 4-20(a)所示是一幅二值图，图 4-20(b)所示是结构元素。用图 4-20(b)来腐蚀图 4-20(a)，将第一次腐蚀所剩下的像素标为 2，就得到图 4-20(c)。

继续腐蚀，将第二次腐蚀所剩下的像素标为 3 就得到图 4-20(d)。

如果继续腐蚀将除去所有像素，因此停止腐蚀。此时综合前面各次腐蚀的结果并对每个像素保留最大值就得到如图 4-20(e)的距离图。用灰度值表示距离图如图 4-20(f)所示，灰度越大代表像素灰度值越大。

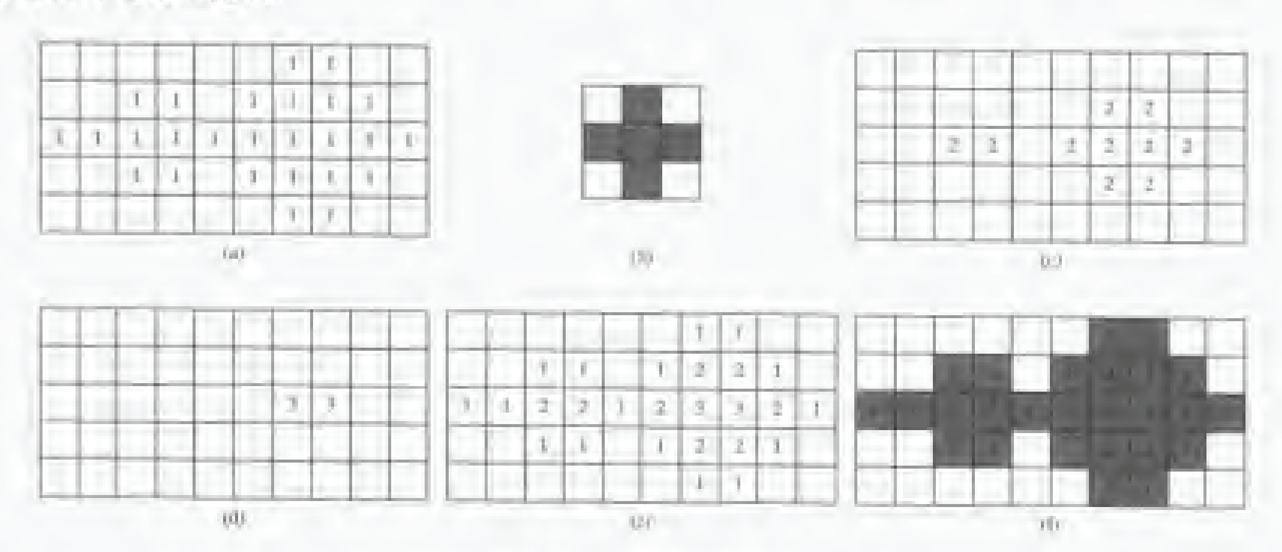


图 4-20 距离图计算示例

从拓扑的角度来看，把距离图当作山脉，则其中的最大值对应山峰而最小值对应山谷。这些山谷就是水线。将各山峰周围的水线连起来可得到对目标的分割。

（2）计算最终腐蚀的集合

在条件 X（X 可看作是一个限定集合）的情况下用 B 膨胀 A，记为“$A\oplus B;X$”，并作如下定义：

$$A\oplus B;\quad X=(A\oplus B)\cap X \tag{4-59}$$

反复条件膨胀是上述的扩展，可记为 $A\oplus\{B\}$：X（这里$\{B\}$代表迭代地用 B 膨胀 A 直到不再有变化）。

最终腐蚀的意思是指反复腐蚀一个目标直到它消失，此时保留这之前最后一步的结果（这个结果也称为目标的种子）。令 $A_k=A\Theta kB$，其中 B 是单位圆，kB 是半径为 k 的圆。最终腐蚀 Y_t 可定义为 A_t 中的元素，如果 $t>k$，则 A_k 在 A_j 中消失。最终腐蚀的第一步是：

$$U_k=(A_{k+1}\oplus\{B\});A_k \tag{4-60}$$

最终腐蚀的第二步是从 A 的腐蚀中减去上述膨胀结果：

$$Y_k=A_k-U_k \tag{4-61}$$

如果图像中有多个目标，可求它们各自 Y_k 的并集就得到最终腐蚀了的目标集合 A。换句话说，最终腐蚀图像是：

$$Y=\bigcup_{k=1,m}Y_k \tag{4-62}$$

其中 m 是腐蚀的次数。

从图 4-20(e)可看出种子就是图中的山峰区域。这些山峰很容易被认出来，因为它们周围都被较小距离的像素所包围。

（3）从种子开始生长回原尺寸但不使各区域相连

这里使用条件粗化，条件粗化如下：

```
Wm=Ym
Do  n=m-1  to  n=0
Wn=Yn∪Wn+1
    Wn=Wn⊙{T^k}, An
    N=n-1
End do
```

其中最后得到的 W_1 是水线运算的结果，m 是对应于前面的腐蚀次数，⊙代表粗化，T^k，k=1,2,⋯,12 代表图 4-21 中的 12 个结构元素。

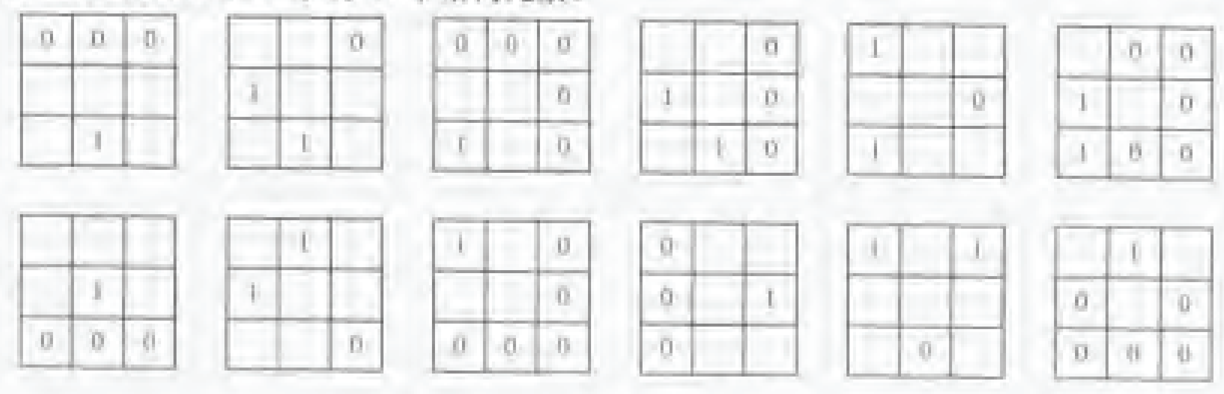


图 4-21　用于水线区域分割的模板

如图 4-22 所示给出一个结合图 4-20 的例子。首先将图 4-20 中的最后两个种子结合（如图 4-22(a)所示）。接下来依次使用图 4-21 的 12 个模板，将各个模板的中心与包围种子的像素重合，如果模板中的 0 和 1 与所重合区域的 0 和 1 匹配，则将该像素赋为 1，即种子区域在尺寸上增加一个像素。图 4-22(b)给出对图 4-22(a)进行一次粗化的结果。在对 12 个结构元素 T^k 都使用后，求 W_n 和一幅条件图像的并集。这样所得的结果是将一个像素宽的区域加在种子周围，除非另一个种子区域与它非常接近。图 4-22(c)给出求图 4-22(b)和图 4-20(c)并集的结果。

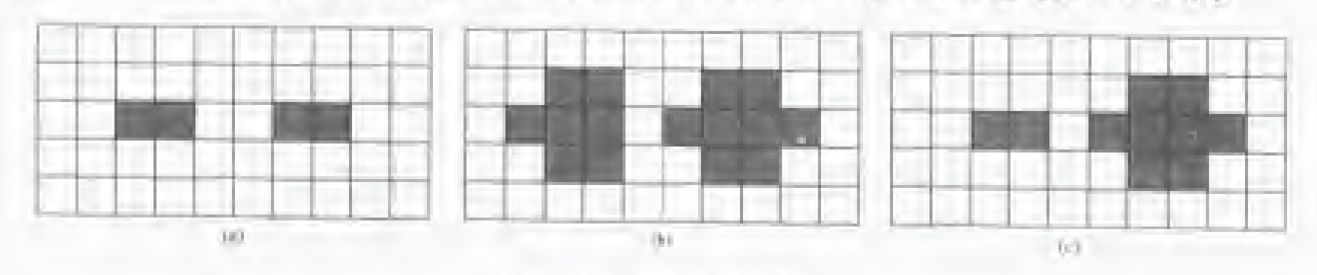


图 4-22　种子生长过程

上述生长和求 4 并集步骤从 T^1 开始进行直到结果没有变化为止。各对应步骤得到的最终腐蚀了的集合，在限制生长（仅包括在对应腐蚀步骤中）的条件下加起来。

4.6.2 借助于统计模式识别方法的分割技术

模式可定义为对图像中的目标或其他感兴趣部分的定量或结构化的描述。通常一个模式可看作是由一个或多个模式符，也可叫特征组成（或排列成）的。图像中不同的区域具有不同的特性，可看作不同的模式。一个模式类则是一组具有某些共同特性的模式。一般常将模式类用 $w_1,w_2,\cdots,w_M$ 表示，其中 M 是类的个数。计算机模式识别包括用一系列自动技术将给定模式赋予它们所属的类。

模式识别的目的是将不同的模式分类，将目标从背景中分割出来也可看作将不同的区域区分出来，这里首要的是确定描述区域特性的参数（特征），对每个区域都可确定一个表征区域的特征矢量。

通过分割，图像被分成为一系列区域，感兴趣的目标被提取了出来。借助特征测量，可以获得描述目标各个特征的测量结果。统计模式识别就是要根据这些测量结果来将各个目标分类，也就是要把各个目标和它们的特征集中到“最为接近”的类别。这里要注意，一方面分割是统计模式识别的基础，对目标分类是建立在分割后提取目标之上的。另一方面，借助统计模式识别中对目标分类的技术也可以进行图像分割。

图像分割可以看作以像素为基元的识别，先确定像素特征如像素灰度值、梯度值等，然后设计分类准则（如以像素灰度作为特征，可借助阈值进行分类，即阈值分割），最后设置分类参数进行分割。

有关具体的统计模式识别将在后续章节继续介绍，相关的识别方法可以用于图像的分割。

第 5 章　特征提取

图像处理的高级阶段是数字图像分析（也可以称为图像理解），主要使用计算机系统，从图像中提取有用的数据或信息，生成非图像的描述或表示，如数值、符号等，即抽取图像特征，从而识别视觉图像。为了能让计算机系统认识图像，人们首先必须寻找出算法，分析图像的特征，然后将其特征用数学的办法表示出来并教会计算机也能懂得这些特征。这样，计算机才能具有认识或者识别图像的本领，称图像模式识别，也叫图像识别。要使计算机具有识别的本领，首先要得到图像的各种特征，像特征提取。

图像特征是指图像的原始特性或属性。每一幅图像都有其本身的特征，其中有些是视觉直接感受到的自然特征，如亮度、边缘的轮廓、纹理或色彩等；有些是需要通过变换或测量才能得到的人为特征，如谱、直方图等。

图像特征提取的结果给出了某一具体的图像中与其他图像相区别的特征。例如，描述物体表面灰度变化的纹理特征，描述物体外形的形状特征等。本章主要介绍如何选择图像特征，以及如何提取这些特征。

5.1　基本概念

5.1.1　问题的提出

图像识别是根据一定的图像特征进行的，显然这些特征的选择很重要，它强烈地影响到图像识别分类器的设计、性能及其识别结果的准确性。特征选择错误，分类就不能分得准确，甚至无法分类。因此，特征选择是图像识别中的一个关键问题。由于实际问题中常常不容易找到那些最重要的特征，或者某些图像特征还会随着环境的变化而发生变化，这就使得特征的选择和提取复杂化。

特征选择和提取的基本任务是如何从众多特征中找出那些最有效的特征。在样本数量不是很多的情况下，用很多特征进行分类器设计，从计算复杂程度和分类器性能来看都是不适宜的。因此研究如何把高维特征空间压缩到低维特征空间以便有效地识别图像成为一个重要的课题，例如在手写体文字识别的特征选择的研究已将近半个世纪，但依然是一个研究的难点和热点。

5.1.2　一些基本概念

例如，通过摄像机把一个物体转换为一个二维灰度阵列，即灰度图像。显然在大多数情况下，不能直接在测量空间中进行识别或设计分类器。一方面是因为测量空间的维数很高（一个 256×256 灰度阵列图像相当于 256×256 维测量空间中的一个点），不便进行识别，更重要的

是这样一种描述并不能直接反映图像的本质，并且它随摄像机位置、光照等因素变化而变化。因此，为了进行识别，需要把图像从测量空间变换到维数大大减少的特征空间，被识别的图像在这个特征空间中就由一个特征向量来表示。为了方便起见，对几个经常用到的有关名词作一些说明。

1. 特征形成

根据待识别的图像，通过计算产生一组原始特征，称之为特征形成。

2. 特征提取

原始特征的数量很大，或者说图像样本是处于一个高维空间中，通过映射（或变换）的方法可以用低维空间来表示样本，这个过程叫特征提取。映射后的特征是原始特征的某种组合。所谓特征提取在广义上就是指一种变换。

3. 特征选择

从一组特征中挑选出一些最有效的特征以达到降低特征空间维数的目的，这个过程叫特征选择。

目前，几乎还没有解析的方法能够指导特征的选择，很多情况下，凭直觉的引导可以列出一些可能的特征表，然后用特征排序方法计算不同特征的识别效率。利用其结果对表进行删减，从而选出若干最好的特征。

良好的特征应具有以下 4 个特点：

（1）可区别性。对于属于不同类的图像来说，它们的特征应具有明显的差异。例如，第 9 章 AGV 自导引小车的路径识别中，路径的颜色是一个好特征，因为它和背景路面的颜色有着显著区别。

（2）可靠性。对于同类的图像，特征值应该比较接近。例如，杂志封面的文字图像分割中，颜色是一个不好的特征。因为，封面文字的颜色可以是各种色彩，尽管它们都属于文字图像。

（3）独立性好。所选择的特征之间彼此不相关。例如细胞的直径和细胞的面积高度相关，因为面积大致与直径的平方成正比。这两个特征基本上反映的相同的属性，即细胞的大小。但是，有时相关性很高的特征组合起来可以减少噪声干扰，它们一般不作为单独的特征使用。

（4）数量少。图像识别系统的复杂程度随着系统维数（特征的个数）迅速增长。尤为重要的是用来训练分类器和测试结果的图像样本随特征数量呈指数关系增长。而且，增加带噪声的特征或与现存特征相关性高的特征实际上会使识别系统的性能下降。

实际应用中特征提取过程往往包括：先测试一组直觉上合理的特征，然后减少成数目合适的满意集。通常符合上述要求的理想特征是很少甚至没有的。

以细胞自动识别为例，通过图像输入得到一批包括正常及异常细胞的数字图像，现在要在这些图像中区分哪些细胞是正常的，哪些是异常的。首先要找出一组能代表细胞性质的特征。为此可以计算细胞的周长、面积、圆度、矩形度、细胞颜色等，这样可以得到很多的细胞原始特征，这一过程就是特征的形成。这样产生的特征很多（几十甚至上百），或者说原始特征的空间的维数很高，需要压缩维数以便识别。一种方式是用映射的方法把原始特征变换为较少的新特征，这就是特征提取。另一种方式就是从原始特征中挑选出一些最具有代表性的特征，这就是特征选择。最简单的特征选择方法就是挑选那些对识别或分类最有影响的特征，另一个方法则是用数学的方法进行筛选比较，找出最有识别或分类信息的特征。本书将重点讨论图像特征的提取，至于特征的形成与具体的图像有较大关系，这里不作介绍。

5.2　纹理特征提取

纹理是图像分析和识别中常用的概念，但目前尚无对它正式的（或者说尚无一致的）定义，一般说来可以认为纹理是由许多相互接近的、互相编织的元素构成，并常富有周期性。也可以可认为是灰度（颜色）在空间以一定的形式变化而产生的图案（模式），它是真实图像固有的特征之一。任何物体的表面，如果一直放大下去进行观察的话一定会显现出纹理，这些纹理特征反映了图像本身的属性，因此利用纹理特征可以将图像中不同目标区别开来。例如沙漠的图像和森林的图像有着不同的空间特征，沙漠的色调或灰度变化较慢，而森林的色调变化较快，通过观察和分析不同物体的图像，可以抽取出构成纹理特征的两个要素：

（1）纹理基元：纹理基元是一种或多种图像基元的组合。纹理基元有一定的形状和大小，例如花布的花纹。

（2）纹理基元的排列组合：基元排列的疏密、周期性、方向性等的不同，能使图像的外观产生极大的改变。例如在植物长势分析中，即使是同类植物，由于地形的不同，生长条件及环境的不同，植物散布形式亦有不同。反映在图像上就是纹理的粗细（植物生长的稀疏）、走向（如靠阳、水的地段应有生长茂盛的植被）等特征的描述和解释。

纹理特征提取指的是通过一定的图像处理技术抽取出纹理特征，从而获得纹理的定量或定性描述的处理过程。因此，纹理特征提取应包括两方面的内容：检测出纹理基元和获得有关纹理基元排列分布方式的信息。在不知道纹理基元或尚未检测出基元的情况下，只能从最小基元——像素开始建立纹理特征模型，这种方式称为文理特征的模型分析。纹理特征分析也可以在已知基元的情况下进行，这种方式称为文理特征的结构分析。

纹理特征的分析方法，大致分为统计方法、结构方法和频谱法。统计方法适用于分析像木纹、森林、山脉、草地那样的纹理细而且不规则的物体；结构方法则适用于象布料的印刷图案或砖花样等一类纹理基元排列较规则的图像；频谱法借助于傅立叶频谱的频率特性来描述周期性的或者近乎周期性的 2-D 图像模式的方向性。本节着重介绍统计方法中几种最常用的方法。

5.2.1　直方图统计特征

前面已经论述过，纹理是像素灰度级变化具有的空间规律性的视觉表现。也就是说，有纹理的区域像素灰度级分布具有一定的形式。通过研究图像中像素的灰度级分布，如直方图或像素的其他性质分布，如边缘的方向直方图，从而建立直方图与纹理基元之间的对应关系，这种方法提取的纹理特征称为直方图统计特征。实际应用中必须注意，这种对应关系是多对一的。

1. 灰度级的直方图特征

直方图是图像窗口中，多种不同灰度的像素分布的概率统计。视觉系统所观察到的图像窗口中的纹理基元必然对应于一定概率分布的直方图，其间存在着一定的对应关系。根据这个特点，就可以让计算机来进行两个适当大小的图像窗口的纹理基元的计算和分析。若已知两个图像窗口中的一个窗口里的纹理基元，且两个窗口的直方图相同或相似，则说明第二个窗口中可能具有类似第一个窗口的纹理基元。若将连续的图像窗口的直方图的相似性进行比较，就可以发现及鉴别纹理基元排列的周期性及紧密性等。具体步骤如下：

（1）选择合适的邻域大小；

（2）对每一个像素，计算出其邻域中的灰度直方图；

（3）比较求出的直方图与已知的各种纹理基元或含有纹理基元的邻域的直方图间的相似性，若相似，则说明图像中可能存在已知的纹理基元；

（4）比较不同像素所对应的直方图间的相似性，从中可以发现纹理基元排列的周期性、疏密性等特征。

上述分析中，最重要的是如何衡量直方图间的相似性，常用的几种相似性度量为：

（1）直方图的均值

设 $h_1(z)$和 $h_2(z)$为两个区域的灰度直方图。

定义：

$$m_1 = \frac{\sum z h_1(z)}{\sum h_1(z)}$$
$$m_2 = \frac{\sum z h_2(z)}{\sum h_2(z)} \qquad (5\text{-}1)$$

若 m_1 和 m_2 充分接近，则说明 $h_1(z)$和 $h_2(z)$是相似的。

（2）直方图的方差

定义：

$$\sigma_1^{\ 2} = \frac{\sum (z-m_1)^2 h_1(z)}{\sum h_1(z)}$$
$$\sigma_2^{\ 2} = \frac{\sum (z-m_2)^2 h_2(z)}{\sum h_2(z)} \qquad (5\text{-}2)$$

如果 m_1 和 m_2 充分接近，并且σ_1^2 和σ_2^2 充分接近，则说明 $h_1(z)$和 $h_2(z)$是相似的。

（3）Kolmogorov—Smimov 检测

定义：

$$H(z) = \int_0^z h(x)dx \qquad (5\text{-}3)$$

则 Kolmogorov—Smimov 检测量定义为：

$$KS = \max_z \left| H_1(z) - H_2(z) \right| \qquad (5\text{-}4)$$

Smoothed—Difference 检测量定义为：

$$SD = \sum_z \left| h_1(z) - h_2(z) \right| \qquad (5\text{-}5)$$

若[*KS*,*SD*]在一个阈值内，就认为 $h_1(z)$和 $h_2(z)$是相似的。

根据上面的分析，可以看出基于灰度级的直方图特征并不能建立特征与纹理基元的一一对应关系。因为直方图是一维信息，不能反映图像纹理的二维灰度变化，即相同的纹理单元具有相同的直方图，但相同的直方图可能会有不同的纹理基元相对应。如图 5-1 所示的两幅不同的纹理图像，它们的灰度直方图是一样的。另一个造成多对一对应关系的因素在于进行直方图相似性度量，即使是不一样或不相似的直方图，也完全可能具有相同的数字特征，如均值或方差。因此，运用直方图纹理特征进行识别图像时，往往还要加上其他特征。



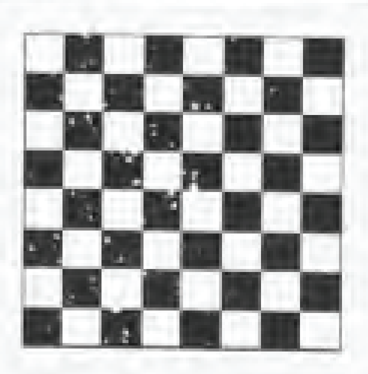


图 5-1　灰度直方图相同的两种纹理

2. 边缘方向直方图

鉴于灰度直方图不能反映图像的二维灰度变化，一个可行的方案是利用边缘方向、大小等的统计性质。例如，先利用微分算子求得图像边缘，然后作出关于边缘的大小和方向的直方图，再对这些直方图的相似性进行度量。另外，单纯地分析边缘方向的直方图（既不进行相似性度量）也可以得到有关纹理特征的一些信息。例如，如果关于边缘方向的直方图在某个范围内具有尖峰，那么就可以知道纹理所具有的对应于这个尖峰的方向性。利用这种方向性，就可以容易地识别图 5-1 中的两种纹理。下面介绍一种边缘方向直方图方法，既构造图像灰度梯度方向矩阵。

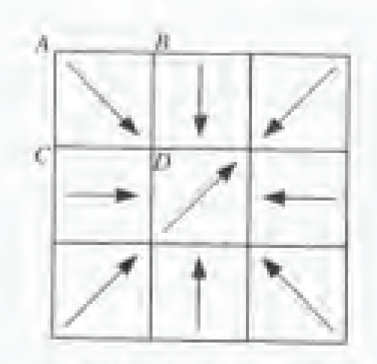


图 5-2　图像的灰度梯度方向矩阵的构造

图像灰度梯度方向矩阵的原理如图 5-2 所示。每一矩阵覆盖有 16 个图像像素，每 4 个像素组成一个小单元。首先计算每一个小单元的梯度，确定其方向。如对图中由 A、B、C 和 D 4 个像素组成的小单元，按下述方法计算 8 个方向的差分值：

$$G_0 = f(A) + f(B) - [f(C) + f(D)]$$

$$G_1 = \sqrt{2}[f(B) - f(C)]$$

$$G_2 = f(B) + f(D) - [f(A) + f(C)]$$

$$G_3 = \sqrt{2}[f(D) - f(A)] \tag{5-6}$$

$$G_4 = -G_0$$

$$G_5 = -G_1$$

$$G_6 = -G_2$$

$$G_7 = -G_3$$

其中 G_0 是指向上方的垂直方向，其余各方向按顺时针方向旋转减少，间隔 45°，即 G_1 对应 45°，G_2 对应 0°……。取 G_0～G_7 中最大值的方向作为该小单元的梯度方向。在计算完这 9 个小单元的方向后，就可计算这个 16 像素的图像区域中所有不同梯度方向的数目，此数目越大，说明图像的内容越离散无规则。用此 16 像素的图像区域为模板，在整个图像上平移计算，并进行分类，就可得出整个图像的灰度梯度方向，就是整个图像的纹理信息，包括纹理走向和纹理疏密等。

5.2.2 图像的自相关函数

定义图像{$f(i,j)$；i,j=0,1,2,…,N-1}的自相关函数为：

$$\rho(x,y)=\frac{\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}f(i,j)f(i+x,j+y)}{\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}f^2(i,j)} \tag{5-7}$$

如果 $i+x>N-1$ 或 $j+y>N-1$，那么定义 $f(i+x,j+y)=0$。所以，$\rho(x,y)$ 也可以看成一幅图像，大小为 $N\times N$。

利用自相关函数 $\rho(x,y)$ 随 x、y 大小而变化的规律，可以描述图像的文理特征。

定义 $d=\sqrt{x^2+y^2}$，则 $x=0$，$y=0$ 时 $\rho(x,y)=1$ 达到最大值。如果图像的纹理较粗，则 $\rho(x,y)$ 随 d 增加而下降的速度较慢；如果图像的纹理较细，则 $\rho(x,y)$ 随 d 增加而下降的速度较快。随着 d 的继续增加，$\rho(x,y)$ 会呈现某种周期性变化，这种周期性可以反映纹理基元的排列规则，例如稀疏、稠密程度等。

通过观察 $\rho(x,y)$ 随哪个方向变化最慢，就可知道纹理基元有很大的可能是沿着这个方向排列的。这可通过对{$-\rho(x,y)$；x,y=0,1,2,…,N-1}的每一点求梯度方向来得到变化最慢的方向。

5.2.3 灰度共生矩阵

纹理是由灰度分布在空间位置上反复交替变化而形成的，因而在图像空间中相隔某距离的两像素间会存在一定的灰度关系，这种关系被称为是图像中灰度的空间相关特性。在灰度直方图中，因为各个像素的灰度是独立地进行处理的，所以不能很好地反映纹理中灰度级空间相关性的规律。故需要研究图像中两个像素灰度级的联合分布的统计形式。通过这种方式得到的纹理特征称为二次统计量（Second Order Statistics）。上述思想正是灰度共生矩阵（Co-Occurence）的基础。

1．基本原理

灰度直方图是对图像上单个像素具有某个灰度进行统计的结果，而灰度共生矩阵是对图像上保持某距离的两像素分别具有某灰度的状况进行统计得到的。两者的最大差别在于，前者表示了单个像素的灰度分布概貌，后者描述了成对像素的灰度组合分布，相对前者来说，后者可看成是两个灰度组合的联合直方图。

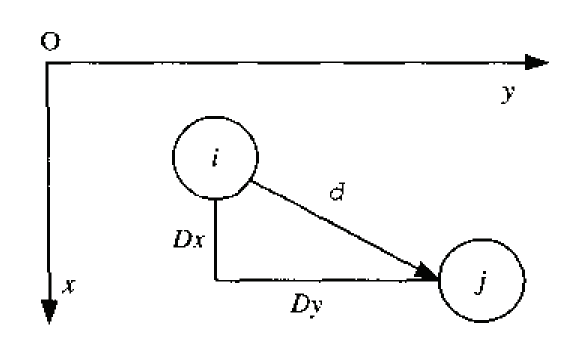


图 5-3　两像素间的位置关系

灰度共生矩阵被定义为从灰度为 i 的点离开某个固定位置关系 $\delta=(D_x,D_y)$ 的点上的灰度为 j 的概率如图 5-3 所示。灰度共生矩阵用 P_δ 表示：

$$P_\delta(i,j),\quad (i,j=0,1,2,\cdots L-1) \tag{5-8}$$

上式中，L 表示图像的灰度级；i，j 分别表示两个像素的灰度；δ 表示两个像素间的空间位置关系。不同的 δ 决定了两个像素间的距离和方向，常用如图 5-4 所示的 4 个方向上位置关系。

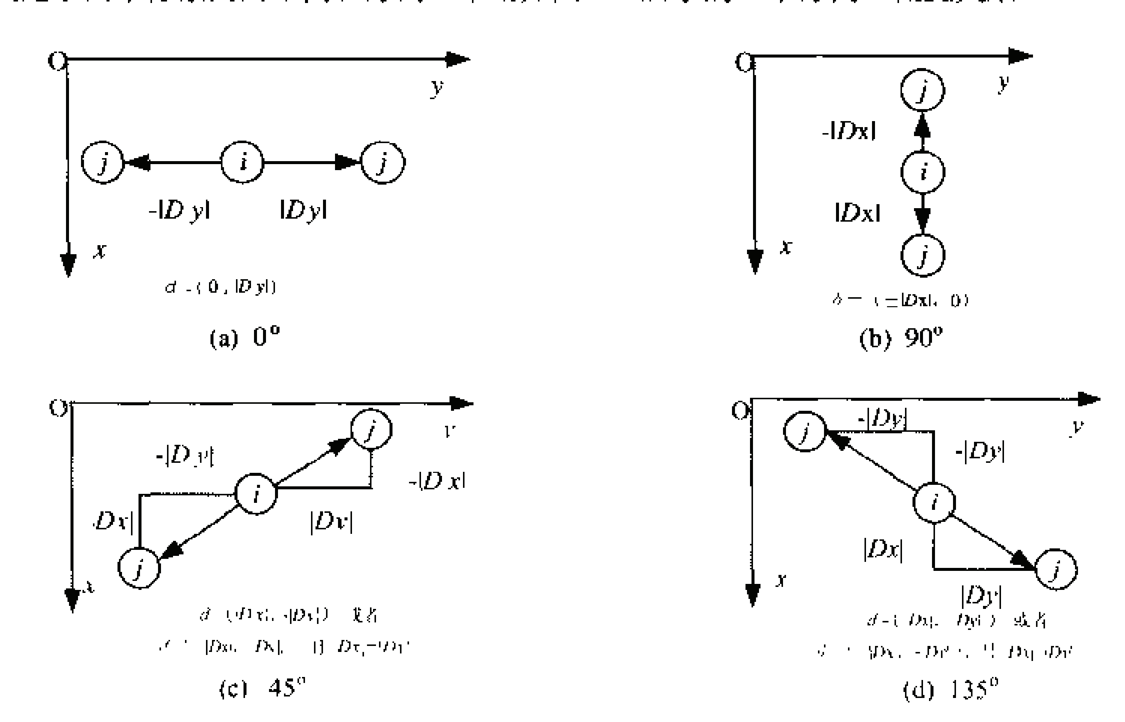


图 5-4　δ 常用的 4 种方向的位置关系

当两像素间的位置关系 δ 选定后，就生成一定 δ 下的灰度共生矩阵 P_δ：

$$P_\delta = \begin{bmatrix} P_\delta(0,0) & P_\delta(0,1) & \cdots & P_\delta(0,j) & \cdots & P_\delta(0,L-1) \\ P_\delta(1,0) & P_\delta(1,1) & \cdots & P_\delta(1,j) & \cdots & P_\delta(1,L-1) \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ P_\delta(i,0) & \cdots & \cdots & P_\delta(i,j) & \cdots & P_\delta(i,L-1) \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ P_\delta(L-1,0) & \cdots & \cdots & P_\delta(L-1,j) & \cdots & P_\delta(L-1,L-1) \end{bmatrix} \tag{5-9}$$

共生矩阵中的一个元素表示了一种灰度组合下出现的次数。如元素产 $P_\delta(1,0)$表示了图像上位置关系为δ的两像素灰度分别为 1 和 0 的情况出现的次数。显然，不同的位置关系（距离或方向），元素值就不相同。灰度矩阵的生成方法参考例 5-1。

【例 5-1】对 5-5 的纹理图像，在不同位置关系下的生成灰度共生矩阵。

图 5-5(a)和图 5-5(b)是两种不同特点的图像纹理，现以常用的 4 个方向的位置关系计算各自的灰度共生矩阵。

0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3

(a) 纹理A

0	1	2	3	0	1	2	3
1	2	3	0	1	2	3	0
2	3	0	1	2	3	0	1
3	0	1	2	3	0	1	2
0	1	2	3	0	1	2	3
1	2	3	0	1	2	3	0
2	3	0	1	2	3	0	1
3	0	1	2	3	0	1	2

(b) 纹理B

图 5-5　纹理图像

（1）0° 方向（水平方向）

位置关系为水平方向，如图 5-4(a)，即 $D_x=0$，令$|D_y|=1$，则$\delta=(0,\pm1)$。若统计 $P_\delta(0,0)$值，就是指位置关系分别为$\delta=(0,1)$和$\delta=(0,-1)$的两像素灰度都为 0 出现的次数之和。$\delta=(0,1)$表示了某像素与其右像素的位置关系；$\delta=(0,-1)$表示了某像素与其左像素的位置关系。也就是说，$P_\delta(0,0)$等于某像素和其右像素灰度都为 0 出现的次数（图 5-5(a)中为 8）加上某像素与其左像素灰度都为 0 出现的次数（图 5-5(a)中为 8）之和。对于图 5-5 中的纹理 *A*，$P_\delta(0,0)=8+8=16$；对于纹理 *B*，$P_\delta(0,0)=0$。若统计 $P_\delta(0,1)$值，纹理 *A* 中某像素与右像素灰度分别为 0 和 1 的状况出现次数为 8，而某像素与左像素灰度分别为 0 和 1 的状况出现次数为 0，所以纹理 *A* 的 $P_\delta(0,1)$为 8；而纹理 *B* 的 $P_\delta(0,1)=14$。这就得到了$\delta=(0,\pm1)$位置关系的灰度共生矩阵。

$$P_\delta = \begin{bmatrix} 16 & 8 & 0 & 0 \\ 8 & 16 & 8 & 0 \\ 0 & 8 & 16 & 8 \\ 0 & 0 & 8 & 16 \end{bmatrix}$$

纹理 *A* 的灰度共生矩阵

$$P_\delta = \begin{bmatrix} 0 & 14 & 0 & 14 \\ 14 & 0 & 14 & 0 \\ 0 & 14 & 0 & 14 \\ 14 & 0 & 14 & 0 \end{bmatrix}$$

纹理 *B* 的灰度共生矩阵

（2）90°方向（垂直方向）

位置关系为垂直方向，如图 5-4(b)所示，既 $D_y=0$，令$|D_x|=1$，则$\delta=(\pm 1,0)$。位置关系$\delta=(\pm 1,0)$的像素实际上就是指某像素与下像素和某像素与上像素两种像素对，这种位置关系下的灰度共生矩阵为：

$$P_\delta=\begin{bmatrix}28&0&0&0\\0&28&0&0\\0&0&28&0\\0&0&0&28\end{bmatrix}\qquad P_\delta=\begin{bmatrix}0&14&0&14\\14&0&14&0\\0&14&0&14\\14&0&14&0\end{bmatrix}$$

纹理 A 的灰度共生矩阵　　　　纹理 B 的灰度共生矩阵

（3）45°方向

位置关系如图 5-4(c)所示，令$|D_x|=|D_y|=1$，则$\delta=(1,-1)$和$\delta=(-1,1)$。这种位置关系下的灰度共生矩阵为：

$$P_\delta=\begin{bmatrix}14&7&0&0\\7&14&7&0\\0&7&14&7\\0&0&7&14\end{bmatrix}\qquad P_\delta=\begin{bmatrix}24&0&0&0\\0&24&0&0\\0&0&24&0\\0&0&0&26\end{bmatrix}$$

纹理 A 的灰度共生矩阵　　　　纹理 B 的灰度共生矩阵

（4）45°方向

位置关系如图 5-4(d)所示，令$|D_x|=|D_y|=1$，则$\delta=(1,1)$和$\delta=(-1,1)$。这种位置关系下的灰度共生矩阵为：

$$P_\delta=\begin{bmatrix}14&7&0&0\\7&14&7&0\\0&7&14&7\\0&0&7&14\end{bmatrix}\qquad P_\delta=\begin{bmatrix}0&0&25&0\\0&0&0&24\\25&0&0&0\\0&24&0&0\end{bmatrix}$$

纹理 A 的灰度共生矩阵　　　　纹理 B 的灰度共生矩阵

2. 灰度共生矩阵的特点

（1）矩阵大小

灰度共生矩阵的行和列分别是两个像素的灰度级，设图像灰度级数为 L，则矩阵为 L 行 L 列共 $L\times L$ 个元素，表示两个像素的灰度组合有 L^2 种。若 $L=256=2^8$，则矩阵的元素数为 2^{16} 个，这么大的矩阵必然使运算量剧增。因此一般先通过灰度直方图的统计分析，在不影响纹理分析的情况下，将灰度适当变换以减少灰度级数，然后再求共生矩阵。

（2）归一化

为了分析方便，矩阵元素常用概率值表示，即将各元素 $P_\delta(i,j)$除以各元素之和 S，得到各

元素都小于1的归一化值 $\hat{P}_\delta(i,j)$，即：

$$\hat{P}_\delta(i,j)=\frac{P_\delta(i,j)}{S} \tag{5-10}$$

由此得到的共生矩阵称为归一化共生矩阵。灰度共生矩阵中各元素之和 S 表示了图像上一定位置关系下像素对的总组合数，对于确定的位置关系 δ，像素对总组合数是一个常数。若图像大小为 $M\times N$，当 $\delta=(0,\pm1)$ 时，每一行形成的像素对组合数为 $2\times(N-1)$，M 行的像素对总组合数为 $S=2M(N-1)$。图 5-5 纹理图像在 0° 方向位置关系下的灰度共生矩阵的归一化表示为：

$$\hat{P}_\delta=\begin{bmatrix}1/7 & 1/14 & 0 & 0\\ 1/14 & 1/7 & 1/14 & 0\\ 0 & 1/14 & 1/7 & 1/14\\ 0 & 0 & 1/14 & 1/7\end{bmatrix} \qquad \hat{P}_\delta=\begin{bmatrix}0 & 1/8 & 0 & 1/8\\ 1/8 & 0 & 1/8 & 0\\ 0 & 1/8 & 0 & 1/8\\ 1/8 & 0 & 1/8 & 0\end{bmatrix}$$

纹理 A 的归一化灰度共生矩阵　　　　纹理 B 的归一化灰度共生矩阵

（3）对称性

在 $L\times L$ 矩阵中，$i=j$ 的元素连成的线称为主对角线。对于在上述常用的 4 个方向的位置关系下生成的灰度共生矩阵，各元素值必定对称于主对角线，即 $P_\delta(i,j)=P_\delta(j,i)$。共生矩阵中形成对称性是由于在这 4 种方向的位置关系中，每一种方向实际上都包含了两种对称的位置关系，如 0° 方向中，包含了 $\delta=(0,|D_y|)$ 和 $\delta=(0,-|D_y|)$ 两种位置。如果位置关系不是上述情况，则生成的灰度共生矩阵并非一定是对称的。

（4）主对角线元素的作用

灰度共生矩阵中的主对角线上元素是一定位置关系下的两像素间灰度组合出现的次数。由于存在沿纹理方向上相近像素的灰度基本相同，垂直纹理方向上相近像素间有较大灰度差的一般规律。因此，这些主对角线元素的大小有助于判别纹理的方向和粗细，对纹理分析起着重要的作用。如图 5-5 中的两种纹理，纹理 A 为 90° 方向，纹理 B 为 45° 方向，当采用 $|D_x|=1$ 或 0，$|D_y|=1$ 或 0 的 4 种方向位置关系生成共生矩阵时，不难发现，沿着纹理方向的共生矩阵中（纹理 A 的 90° 方向和纹理 B 的 45° 方向）主对角线元素值很大，而其他元素的值全为零，这正说明了沿着纹理方向上没有灰度变化。可见，大的主对角线元素提供了识别纹理方向的可能性，垂直纹理方向（纹理 A 为 0°，纹理 B 为 135°）构成的共生矩阵，对于纹理 B，主对角线元素全为零，说明在垂直纹理的方向上相邻像素的灰度都不相同。那就是说，灰度变化频繁，纹理较细。相对来说，纹理 A 较粗，共生矩阵主对角线上的元素不为零，表明了相邻像素的灰度变化缓慢。

（5）元素值的离散性

灰度共生矩阵中元素值相对于主对角线的分布可用离散性来表示，它常常反映纹理的粗细程度。离开主对角线远的元素的归一化值高，即元素值的离散性大，也就是说，一定位置关系的两像素间灰度差大的比例高。若仍以 $|D_x|=1$ 或 0，$|D_y|=1$ 或 0 的 4 种方向位置关系为例，离散性大意味着相邻像素间灰度差大的比例高，说明图像上垂直于该方向的纹理较细；相反，则图像上垂直于该方向上的纹理较粗。当非主对角线上的元素的归一化值全为零时，元素值的离

散性最小，即图像上垂直于该方向上不可能出现纹理。比较图 5-5 中的纹理 A 的 0° 方向和纹理 B 的 135° 方向灰度共生矩阵的元素值，可知纹理 B 的离散性较纹理 A 的离散性大，因而纹理 A 较粗，纹理 B 较细。

3. 特征参数

从灰度共生矩阵抽取出的纹理特征参数有以下几种。

（1）二阶矩

$$f_1=\sum_{i=0}^{L-1}\sum_{j=0}^{L-1}\hat{P}_\delta^2(i,j) \tag{5-11}$$

二阶矩是图像灰度分布均匀性的度量。当灰度共生矩阵中的元素分布较集中于主对角线时，说明从局部区域观察图像的灰度分布是较均匀的。从图像整体来观察，纹理较粗，此时二阶矩值 f_1 则较大，反过来则二阶矩值 f_1 较小。二阶矩是灰度共生矩阵像素值平方的和，所以，它也称为能量。粗纹理二阶矩值 f_1 较大，可以理解为粗纹理含有较多的能量。细纹理 f_2 较小，也即它含有较少的能量。

（2）对比度

$$f_2=\sum_{n=0}^{L-1}n^2\{\sum_{i=0}^{L-1}\sum_{\substack{j=0\\|i-j|=n}}^{L-1}\hat{P}_\delta(i,j)\} \tag{5-12}$$

图像的对比度可以理解为图像的清晰度，即纹理的清晰程度。在图像中，纹理的沟纹越深，则其对比度 f_2 越大，图像的视觉效果越是清晰。

（3）相关

$$f_3=\frac{\sum_{i=0}^{L-1}\sum_{j=0}^{L-1}ij\hat{P}_\delta(i,j)-\mu_1\mu_2}{\sigma_1^2\sigma_2^2} \tag{5-13}$$

式中 μ_1、μ_2、σ_1 和 σ_2 分别定义为：

$$\mu_1=\sum_{i=0}^{L-1}i\sum_{j=0}^{L-1}\hat{P}_\delta(i,j) \tag{5-14}$$

$$\mu_2=\sum_{j=0}^{L-1}j\sum_{i=0}^{L-1}\hat{P}_\delta(i,j) \tag{5-15}$$

$$\sigma_1{}^2=\sum_{i=0}^{L-1}(i-\mu_1)^2\sum_{j=0}^{L-1}\hat{P}_\delta(i,j) \tag{5-16}$$

$$\sigma_2{}^2=\sum_{j=0}^{L-1}(j-\mu_2)^2\sum_{i=0}^{L-1}\hat{P}_\delta(i,j) \tag{5-17}$$

相关能够用来衡量灰度共生矩阵的元素在行的方向或列的方向的相似程度。例如，某图像具有水平方向的纹理，则图像在 $\theta=0^\circ$ 方向的灰度共生矩阵的相关值 f_3 往往大于 $\theta=45^\circ$，$\theta=90^\circ$

和$\theta=135°$的灰度共生矩阵的相关值f_3。

（4）熵

$$f_4=-\sum_{i=0}^{L-1}\sum_{j=0}^{L-1}\hat{P}_\delta(i,j)\log\hat{P}_\delta(i,j) \tag{5-18}$$

熵值是图像具有的信息量的度量，纹理信息是图像信息的一种。若图像没有任何纹理，则灰度共生矩阵几乎为零阵，熵值f_4接近为0。若图像充满着细纹理，则$\hat{P}_\delta(i,j)$值近似相等，该图像的熵值f_4最大。若图像中分布着较少的纹理，$\hat{P}_\delta(i,j)$的数值差别较大，则该图像的熵值f_4较少。

上述的4个参数为应用灰度共生矩阵进行纹理分析的主要参数，可以组合起来，成为纹理分析的特征参数使用。还有一些从灰度共生矩阵引导出来的参数也可以进行纹理分析。

（5）逆差矩

$$f_5=\sum_{i=0}^{L-1}\sum_{j=0}^{L-1}\frac{\hat{P}_\delta(i,j)}{1+(i-j)^2} \tag{5-19}$$

（6）和平均

$$f_6=\sum_{k=0}^{2L-2}\sum_{i=0}^{L-1}\sum_{\substack{j=0\\i+j=k}}^{L-1}\hat{P}_\delta(i,j) \tag{5-20}$$

（7）和方差

$$f_7=\sum_{k=0}^{2L-2}\sum_{i=0}^{L-1}\sum_{\substack{j=0\\i+j=k}}^{L-1}(i-f_6)^2\hat{P}_\delta(i,j) \tag{5-21}$$

（8）和熵

$$f_8=-\sum_{k=0}^{2L-2}\sum_{i=0}^{L-1}\sum_{\substack{j=0\\i+j=k}}^{L-1}\hat{P}_\delta(i,j)\log\sum_{i=0}^{L-1}\sum_{\substack{j=0\\i+j=k}}^{L-1}\hat{P}_\delta(i,j) \tag{5-22}$$

（9）差平均

$$f_9=\sum_{k=0}^{L-1}\sum_{i=0}^{L-1}\sum_{\substack{j=0\\|i-j|=k}}^{L-1}\hat{P}_\delta(i,j) \tag{5-23}$$

（10）差方差

$$f_{10}=\sum_{k=0}^{L-1}(k-f_9)^2\sum_{i=0}^{L-1}\sum_{\substack{j=0\\|i-j|=k}}^{L-1}\hat{P}_\delta(i,j) \tag{5-24}$$

（11）差熵

$$f_{11}=-\sum_{k=0}^{L-1}\sum_{i=0}^{L-1}\sum_{\substack{j=0\\|i-j|=k}}^{L-1}\hat{P}_\delta(i,j)\log\sum_{i=0}^{L-1}\sum_{\substack{j=0\\|i-j|=k}}^{L-1}\hat{P}_\delta(i,j) \tag{5-25}$$

实际应用中，根据要求和有关图像及纹理的先验知识，选择若干 d 或者确定自己所要求的（D_x,D_y）的选择方式，计算出不同（D_x,D_y）下的共生矩阵，分别导出不同的特征量。把所有的这些度量排列起来就可以完成图像或纹理到数字特征的一一对应关系，进一步识别数字特征即可实现纹理或图像的识别任务。

5.2.4　灰度－梯度共生矩阵

灰度－梯度共生矩阵法前面介绍的灰度级直方图和边缘梯度直方图的结合，它考虑的是像素级灰度和边缘梯度大小的联合统计分布。灰度直方图是一幅图像灰度在图像中分布的最基本统计信息，而图像的梯度检出了图像中的灰度跳变部分。将图像的梯度信息加进灰度共生矩阵，则使得共生矩阵更能包含图像的纹理基元及其排列的信息。

考虑图像$\{f(i,j);\ i,j=0,1,2\cdots,N-1\}$。灰度级数目为 L。对该图像采用第 4 章介绍的微分算子，可以得到其梯度图像$\{g(i,j);\ i,j=0,1,2\cdots,N-1\}$。将此梯度图像进行灰度级离散化，设灰度级数目为 L_g，离散化间隔为 $1/L_g$，即新的灰度级应为：

$$G(i,j)=\frac{g(i,j)-g_{\min}}{g_{\max}-g_{\min}} \tag{5-26}$$

式中 $g_{\min}$，$g_{\max}$ 分别是$\{g(i,j);\ i,j=0,1,2,\cdots,N-1\}$的最小值和最大值。

经此变换后，梯度图像的为$\{G(i,j);\ i,j=0,1,\cdots,N-1\}$。灰度级数目为 L_g。

灰度－梯度共生矩阵定义如下：

$$\left\{H(x,y);\quad x=0,1,\cdots,L-1;\quad y=0,1,\cdots,L_g-1\right\} \tag{5-27}$$

$H(x,y)$定义为集合$\left\{(i,j)\middle|f(i,j)=x\text{且}G(i,j)=y;\quad i,y=0,1\cdots,N-1\right\}$中元素的数目，既灰度为 x，梯度为 y 的总像素点。

对灰度－梯度共生矩阵进行归一化处理，使其各元素之和为 1。通过下式来实现：

$$\hat{H}(x,y)=\frac{H(x,y)}{\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}H(x,y)} \tag{5-28}$$

而$\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}H(x,y)=N\times N=N^2$，所以（5-28）式可以表示为：

$$\hat{H}(x,y)=\frac{H(x,y)}{N^2} \tag{5-29}$$

灰度－梯度共生矩阵常用的数字特征有：

（1）小梯度优势

$$T_1=\frac{\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}\hat{H}(x,y)/(y+1)^2}{\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}\hat{H}(x,y)} \tag{5-30}$$

（2）大梯度优势

$$T_2=\frac{\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}\hat{H}(x,y)y^2}{\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}\hat{H}(x,y)} \tag{5-31}$$

（3）灰度分布的不均匀性

$$T_3=\frac{\sum_{x=0}^{L-1}\left[\sum_{y=0}^{L_g-1}\hat{H}(x,y)\right]^2}{\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}\hat{H}(x,y)} \tag{5-32}$$

（4）梯度分布的不均性

$$T_4=\frac{\sum_{y=0}^{L_g-1}\left[\sum_{x=0}^{L-1}\hat{H}(x,y)\right]^2}{\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}\hat{H}(x,y)} \tag{5-33}$$

（5）能量

$$T_5=\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}\hat{H}^2(x,y) \tag{5-34}$$

（6）灰度平均

$$T_6=\sum_{x=0}^{L-1}x\sum_{y=0}^{L_g-1}\hat{H}(x,y) \tag{5-35}$$

（7）梯度平均

$$T_7=\sum_{y=0}^{L_g-1}y\sum_{x=0}^{L-1}\hat{H}(x,y) \tag{5-36}$$

（8）灰度均方差

$$T_8=\left\{\sum_{x=0}^{L-1}(x-T_6)^2\sum_{y=0}^{L_g-1}\hat{H}(x,y)\right\}^{\frac{1}{2}} \tag{5-37}$$

（9）梯度均方差

$$T_9=\left\{\sum_{y=0}^{L_g-1}(y-T_7)^2\sum_{x=0}^{L-1}\hat{H}(x,y)\right\}^{\frac{1}{2}} \tag{5-38}$$

（10）相关

$$T_{10}=\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}(x-T_6)(y-T_7)\hat{H}(x,y) \tag{5-39}$$

（11）灰度熵

$$T_{11}=-\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}\hat{H}(x,y)\log\sum_{y=0}^{L_g-1}\hat{H}(x,y) \tag{5-40}$$

（12）梯度熵

$$T_{12}=-\sum_{y=0}^{L_g-1}\sum_{x=0}^{L-1}\hat{H}(x,y)\log\sum_{x=0}^{L-1}\hat{H}(x,y) \tag{5-41}$$

（13）混合熵

$$T_{11}=-\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}\hat{H}(x,y)\log\hat{H}(x,y) \tag{5-42}$$

（14）惯性

$$T_{14}=\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}(x-y)^2\,\hat{H}(x,y) \tag{5-43}$$

（15）逆差矩

$$T_{14}=\sum_{x=0}^{L-1}\sum_{y=0}^{L_g-1}\frac{\hat{H}(x,y)}{1+(x-y)^2} \tag{5-44}$$

5.2.5　基于变换的特征

1. 傅立叶变换

在纹理分析中使用傅立叶变换的原因主要是因为图像傅立叶变换的能量谱能在一定程度上反映某些纹理特征，图像$\{f(x,y)\}$的傅里叶变换定义为：

$$F(u,v)=\iint f(x,y)\exp\left[-j2\pi(ux+vy)\right]\mathrm{d}x\mathrm{d}y \tag{5-45}$$

其功率谱定义为：

$$\left|F(u,v)\right|^2=F(u,v)F^*(u,v) \tag{5-46}$$

其中*表示复共轭。

很容易证明，如果一幅图像的纹理较粗糙，即图像的灰度变化很少或较慢，则在小

$\sqrt{u^2+v^2}$ 值处 $|F(u,v)|^2$ 应有较大的值；如果一幅了图像的纹理较细腻，即图像的灰度变化频繁或较快，则在大的 $\sqrt{u^2+v^2}$ 值处 $|F(u,v)|^2$ 应有较大的值。因此，如果想要检测纹理的粗糙、细腻性质，一个有用的度量就是 $|F(u,v)|^2$ 随 $\sqrt{u^2+v^2}$ 变化的情况。将 $|F(u,v)|^2$ 用极坐标表示为 $|F(\rho,\theta)|^2$，$\sqrt{u^2+v^2}$ 就成为ρ。$|F(\rho,\theta)|^2$ 不仅与ρ，而且与θ有关。我们暂时不考虑θ，而用下面的综合性度量来检测纹理的粗糙性：

$$t(\rho)=\int_0^{+\infty}|F(\rho,\theta)|^2\mathrm{d}\theta \tag{5-47}$$

这样，$t(\rho)$的峰的位置反映了纹理构成元素或纹理基元的大小。设ρ_0 是 $t(\rho)$的峰点，ρ_0 越小，说明纹理基元越大，纹理越粗糙；ρ_0 越大，说明纹理基元越小，纹理越细。如果 $t(\rho)$ 没有明显的峰或峰的数目很多，说明图像中纹理杂乱无章，存在有多种尺度上的纹理或空间相关

另一个与能量谱紧密相连的是纹理的方向性。假如一幅图像有一条朝向（与 x 轴夹角）为 θ_0 的边缘，那么沿着与此边缘垂直的方向上，即$\theta_0\pm\pi/2$ 的方向上，有可能观察到明显的纹理。这是因为纹理基元一定是靠边缘将其与其他基元或物体分开的。这样的话，图像的傅里叶变换在(u,v)空间的$\theta_0\pm\pi/2$ 方向上应有较大的分量，即 $|F(\rho,\theta)|^2$ 应在$\theta=\theta_0\pm\pi/2$ 方向上有较大的值。为此，构造如下对方向敏感的度量：

$$t(\theta)=\int|F(\rho,\theta)|^2\mathrm{d}\rho \tag{5-48}$$

如果 $t(\theta)$在θ_0 存在峰点，则说明原图像的纹理很有可能是沿着θ_0 方向延伸的。如果不存在明显的峰点或者峰点过多，则说明纹理无明显方向性或者纹理排列杂乱。

注意：$t(\rho)$是对方向不敏感而对频率敏感（指（u,v）空间的频率），而 $t(\theta)$对频率不敏感但对方向敏感。因而可以发展一些综合这两种度量的方法。

2. 余弦变换

和傅立叶变换相比，余弦变换是实图像到实图像的变换，因而不会向上面介绍的傅立叶变换那样，由于只考虑幅度图像，丢失了相位信息。余弦变换的物理含义非常清晰，它的变换基矢量非常接近一阶 Markov 模型的自相关阵特征向量，因此反映了空间的相关性。对于平坦完全无纹理的区域，其余弦变换 $F(u,v)$只有（0,0）分量既平均灰度。对于纹理粗糙的区域，由于图像具有跨越距离较大的空间相关性，在低频分量即小的 $\sqrt{u^2+v^2}$ 处 $|F(u,v)|^2$ 有较大的值；对于纹理细腻的区域，由于图像具有较小的空间相关性，在高频分量即大的 $\sqrt{u^2+v^2}$ 处 $|F(u,v)|^2$ 有较大的值。同样，也可分析纹理的朝向与 $F(u,v)$的关系。据此可得到由 $F(u,v)$导出的纹理特征度量。有兴趣读者可参考有关文献。

本节分析的纹理特征中没有指明纹理基元到底是什么，所给出的文理特征多是基元排列的性质，如疏密、朝向等的定性描述，进一步的确定涉及到对抽取出的数字特征或一些特征度量

进行识别。有关的这些特征提取需要根据具体的识别图像、识别策略等加以决定。

5.3　形状和结构特征提取

对计算机图像识别系统而言，物体的形状是一个赖以识别的重要特征。一个图像形状和结构特征有两种形式，一种是数字特征主要包括几何属性（如长短、面积、距离和凹凸特性等），统计属性（如黑色像素点在垂直方向的投影）和拓扑属性（如连通、欧拉数）；另一种是由字符串和图等所表示的句法语言。它可以刻画某一图像不同部分之间的相互关系（如文字识别中的笔划关系），也可以描述不同目标间的关系。

由于感兴趣的是图像的形状和结构特征，所以其灰度信息往往可以忽略，只要能将它与其他目标或背景区分开来即可。常用的一种技术是二值化图像，即将感兴趣的部分（区域或边界）标以最大灰度级，把背景（也包括其他任何不感兴趣的部分）标以最小灰度级，通常为零。二值化图像在形状和结构分析中占有很重要的地位，本节讨论的算法如没有特别说明都是基于二值化图像的。

5.3.1　区域内部的数字特征

1. 矩

给定二维连续函数 f(x,y)，下式定义了其 pq 阶矩：

$$M_{pq}=\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}x^{p}y^{q}f(x,y)\mathrm{d}x\mathrm{d}y \qquad p、q=0,1,2\cdots \tag{5-49}$$

矩在文字识别中作为有效统计特征而被广泛运用，它之所以能被用来表征一幅二维图像是基于下面的帕普利斯（Papoulis）惟一性定理：如果 $f(x,y)$ 是分段连续的，只在 xy 平面的有限部分中有非零值，则所有各阶矩皆存在，并且矩序列 $\{M_{pq}\}$ 惟一地被 $f(x,y)$ 所确定，反之 $\{M_{pq}\}$ 也惟一地确定 $f(x,y)$。

对一副二值图像 $\{f(i,j);\ i,j,=0,1,\cdots,N-1\}$ 来说，上述条件无疑可被满足。因此，可定义其 pq 阶矩为：

$$M_{pq}=\sum\sum f(i,j)i^{p}j^{q} \tag{5-50}$$

不同 p、q 值下可以得到不同的图像矩 M_{pq}，常用的区域矩特征有以下几个：

（1）质心

$$(\bar{i},\bar{j})=(\frac{M_{10}}{M_{00}},\frac{M_{01}}{M_{00}}) \tag{5-51}$$

（2）中心矩

$$m_{pq}=\sum\sum f(i,j)(i-\bar{i})^{p}(j-\bar{j})^{q} \tag{5-52}$$

（3）*Hu* 矩组

Hu 矩组是 $\{m_{pq}\}$ 前 8 个矩的函数，它满足平移、旋转不变性，因而可被广泛地应用于区域

形状识别中。

$$M_1 = m_{20} + m_{02}$$

$$M_2 = (m_{20} - m_{02})^2 + 4m_{11}^2$$

$$M_3 = (m_{30} - 3m_{12})^2 + (3m_{21} + m_{03})^2$$

$$M_4 = (m_{30} + m_{12})^2 + (m_{21} + m_{03})^2$$

$$M_5 = (m_{30} - 3m_{12})(m_{30} + m_{12})\left[(m_{30} + m_{12})^2 - 3(m_{21} + m_{03})^2\right] + (3m_{21} - m_{03})(m_{21} + m_{03})\left[3(m_{30} + m_{12})^2 - (m_{21} + m_{03})^2\right]$$

$$M_6 = (m_{20} - m_{02})\left[(m_{30} + m_{12})^2 - (m_{21} + m_{03})^2\right] + 4m_{11}(m_{30} + m_{12})(m_{21} + m_{03})$$

$$M_7 = (3m_{12} - m_{30})(m_{30} + m_{12})\left[(m_{30} + m_{12})^2 - 3(m_{21} + m_{03})^2\right] + (3m_{21} - m_{03})(m_{21} + m_{03})\left[3(m_{03} + m_{12})^2 - (m_{12} + m_{03})^2\right] \quad (5\text{-}53)$$

如果上述的 7 个 *Hu* 矩中的 m_{pq} 用 $\dfrac{m_{pq}}{m_{00}^{(p+q)/2}}(p+q=2,3,\cdots)$ 来代替，则得到的矩还可以满足尺度不变性。特别地，M_7 满足镜像对称不变性。

（4）面积

区域的面积定义为区域中的像素点数：

$$S = \frac{M_{00}}{\max} \quad (5\text{-}54)$$

其中 max 为二值图像的最大灰度级。

（5）扁度

扁度定义为区域的长短轴之比：

$$e = \frac{m_{20} + m_{02} + \sqrt{(m_{20} + m_{02})^2 - 4m_{20}m_{02} + 4m_{11}^2}}{m_{20} + m_{02} - \sqrt{(m_{20} + m_{02})^2 - 4m_{20}m_{02} + 4m_{11}^2}} \quad (5\text{-}55)$$

根据帕普利斯的定理，需要无穷多的 M_{pq} 序列才能确定 $f(x,y)$。在实际应用中，这是不可能实现的，通常取前几阶矩即可，但是这会带来误差。

2. 投影

投影的示意图如图 5-6 所示。图像函数为$\{f(x,y)\}$，s 为投影方向，t 为与其垂直的方向，t 与 x 轴夹角为θ，则$\{f(x,y)\}$沿着 s 的投影定义为：

$$p(t,\theta)=\int_{-\infty}^{+\infty} f(t\cos\theta-s\sin\theta,t\sin\theta+s\cos\theta)\mathrm{d}s \tag{5-56}$$

当θ固定时，$p(t,\theta)$为 t 的函数，亦即一个一维波形。不断地从 0～2π 变换θ，可得到在不同方向上$\{f(x,y)\}$的投影。

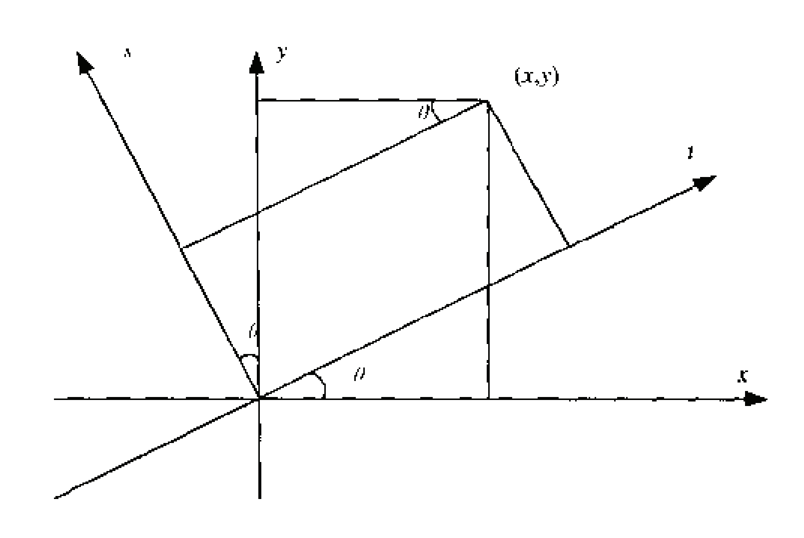


图 5-6　投影坐标(t,s)与原坐标系(x,y)间的对应关系

由投影定理，对满足一定条件的$\{f(x,y)\}$，如果知道全部方向上的$\{p(t,\theta)\}$，就可以惟一地恢复出$\{f(x,y)\}$。然而统矩方法一样，获得所有方向上的投影在实际应用中是行不通的。通常取若干个特定方向上的投影作为$\{f(x,y)\}$的形状特征度量，特别地，在 x 轴和 y 轴上的投影定义为：

$$p_x=p(t,0)=\int_{-\infty}^{+\infty} f(t,s)\mathrm{d}s=\int_{-\infty}^{+\infty} f(x,y)\mathrm{d}y \tag{5-57}$$

$$p_y=p(t,\frac{\pi}{2})=\int_{-\infty}^{+\infty} f(-s,t)\mathrm{d}s=\int_{-\infty}^{+\infty} f(x,y)\mathrm{d}x \tag{5-58}$$

应用投影定理，可以把二维图像的问题转变为一维的曲线波形的问题。

3. 欧拉数

图像的欧拉数是图像的一种拓扑性质度量，它表明了图的连通性。欧拉数定义为一个图中或一个区域中的孔数 H 和连接部分数 C 的差：$E=C-H$。

对数字图像而言，如果图像的背景用 0 标记，目标物体用 1 标记，则欧拉数可用下式计算：

$$E=n(1)-n\begin{pmatrix}1\\1\end{pmatrix}-n\begin{pmatrix}1&1\end{pmatrix}+n\begin{pmatrix}1&1\\1&1\end{pmatrix} \tag{5-59}$$

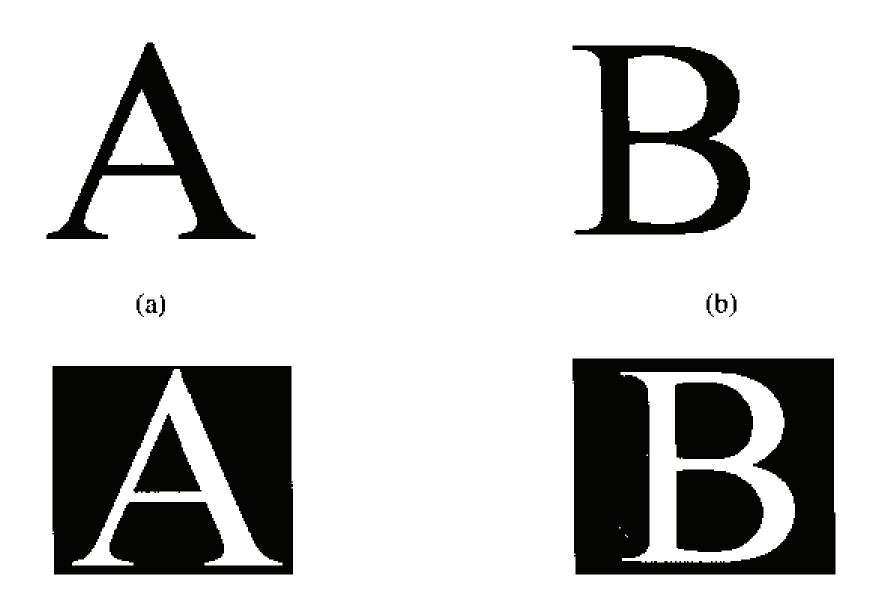
$n(1)$表明图像中像素点的数目，$n\begin{pmatrix}1\\1\end{pmatrix}$表示二值图像中具有垂直相邻两个 1 标记的状态记数，$n\begin{pmatrix}1&1\end{pmatrix}$表示具有水平相邻 1 标记的状态记数，$n\begin{pmatrix}1&1\\1&1\end{pmatrix}$表示 4 个 1 标记相邻的状态记数。

MATLAB 中可以用 bweuler 函数来求图像的欧拉数，调用格式如下：

```
E=bweuler(I,n)
```

其中，I 为输入的二值图像，n 设置邻域的大小，即 4—邻域或 8—邻域，默认值为 8。必须注意，二值图像的背景为 0，目标为 1，检测出的欧拉数才是目标图像的欧拉数。

【例 5-2】计算图 5-7(a)、5-7(b)中的欧拉数



(a) (b)

(c) 用于计算(a)图欧拉数的二值图像 (d) 用于计算(b)图欧拉数的二值图像

图 5-7 图像区域的欧拉数计算

```
I1=imread('5-7(A).jpg'); % 图 5-7(a)的欧拉数计算
BW1=im2bw(I1,0.7); % 图像二值化。原始图像虽然看起来向二值图像，实质为 RGB 图像。
figure,imshow(~BW1);
E1=bweuler(~BW1,8) % 二值化后的图像，目标为 0，背景为 1，必须先求反后计算欧拉数。

I2=imread('5-7(B).jpg'); % 图 5-7(b)的欧拉数计算
BW2=im2bw(I2,0.7);
figure,imshow(~BW2);
E2=bweuler(~BW2,8)
```

计算可得：$E1=0$；图(c)中的连接部分 $C=1$，孔数 $H=1$；$E2=-1$；图(d)中的连接部分 $C=1$，孔数 $H=2$。

4. 几何特征

本书在第 7 章的癌细胞形状分析中充分利用细胞的形状特征，相关的应用例子及算法请读者参考第 7 章。本节作简单的基本概念介绍。

（1）面积和周长

面积 S 和周长 L 是描述区域大小的基本特征。计算图像中某个区域的面积以及该区域的周长，根据它们的比值可以分析或提取该区域所代表的图像形状特征。

粗略地说，图像中的区域面积 S 就是图像中相同标记的像素数目。由于连续图像采用离散的像素点描述时，产生了误差。例如，一个包含 50 个像素的对角线比一个 50 个像素的水平直线要长。因此，在计算面积的过程中对每一个不同像素模式加上不同的权值，以减少误差。

区域的周长 L 用区域中相邻边缘点间的距离之和来表示，同样存在误差补偿的问题。

MATLAB 中可以用函数 bwarea 来计算图像的面积，该函数充分考虑了误差补偿的相关问题。周长则可以用 bwperim 函数提取出图像的周边后，再通过像素点个数计算周长。

（2）圆形度 R

圆形度用来表示目标物体形状接近圆形的程度，其计算公式为：

$$R = \frac{4\pi S}{L^2} \tag{5-60}$$

式中 S 为区域的面积；L 为周长；R 的取值范围为 $0<R\leqslant 1$，R 越大，则区域越接近圆形。以连续的圆形，正方形和正三角形为例，它们的圆形度 R 分别为：圆形 R=1；正方形 R=0.79；正三角形 R=0.60。

（3）凹凸特性

凹凸特性时区域的基本特性之一。区域的凹凸性可以通过以下方法进行判别：区域内任意两像素间的连线穿过去域外的像素，则此区域为凹形。相反，区域内任意两像素间的连线不穿过区域外的像素，则称为凸形。在粘连字符的切分和文字识别等领域，经常利用字符轮廓的凹凸特性分析其特征，读者可以参考提第 8 章的文字识别有关内容。

5.3.2　基于边界的形状特征

1. 傅立叶描绘子

对于边界来说，最重要的是组成边界的点的位置信息。灰度信息完全可以忽略。因此可以将边界看成是直角坐标系下的点集构成的曲线 $y=f(x)$，其中 x 是横坐标，y 是纵坐标。可利用傅立叶变换描述 $y=f(x)$，这一方法称为傅立叶描绘子。

（1）Zahn 描绘子

若以 $y=f(x)$直接进行傅里叶变换，则变换的结果将与具体的 x 和 y 坐标值有关，不能满足平移和旋转的不变性要求。为了解决这个问题，引入封闭曲线本身的内禀参量构造曲线方程，再做傅立叶变换。

由于边界通常是封闭曲线。设 r 是顺时针方向的封闭曲线。引入曲线本身的内禀参量即曲线弧长 l 构造曲线方程，它的参数表达式为

$$z(l) = (x(l), y(l)) \tag{5-61}$$

式中 $0 \le l \le L$，L 是曲线全长。曲线的初始点为 l=0，$\theta(l)$ 是曲线弧长为 l 的点的切线方向。

定义：

$$\varphi(l) = \theta(l) - \theta(0) \tag{5-62}$$

则 $\varphi(l)$ 的变化规律可以描述封闭曲线的形状，很明显它是平移和旋转不变的。由于 $\varphi(l)$ 不是一个周期函数，为将其变换为周期函数引入另一个变量 $t = \frac{2l\pi}{L}$，则 $t \in [0 \quad 2\pi]$。可定义：

$$\varphi^*(t)=\varphi(tL/2\pi)+t,\quad t\in[0\quad 2\pi] \tag{5-63}$$

则$\varphi^*(t)$是[0,2π]上的周期函数，而且它对封闭曲线r的平移、旋转和尺度都是不变的。构造$r\to\varphi^*(t)$间的对应关系是一对一的，即在尺度变化下是相似的，如果反演$\varphi^*(t)\to r$，则可得出一组相似的封闭曲线。

由于$\varphi^*(t)$是周期函数，因此可用它的傅立叶系数来描述它，在[0,2π]上展成傅立叶级数为：

$$\varphi^*(t)=a_0+\sum_{k=1}^{+\infty}(a_k\cos kt+b_k\sin kt) \tag{5-64}$$

其中：

$$a_0=\frac{1}{2\pi}\int_0^{2\pi}\varphi^*(t)dt$$

$$a_n=\frac{1}{\pi}\int_0^{2\pi}\varphi^*(t)\cos nt dt$$

$$b_n=\frac{1}{\pi}\int_0^{2\pi}\varphi^*(t)\sin nt dt$$

其中n=1,2,……。

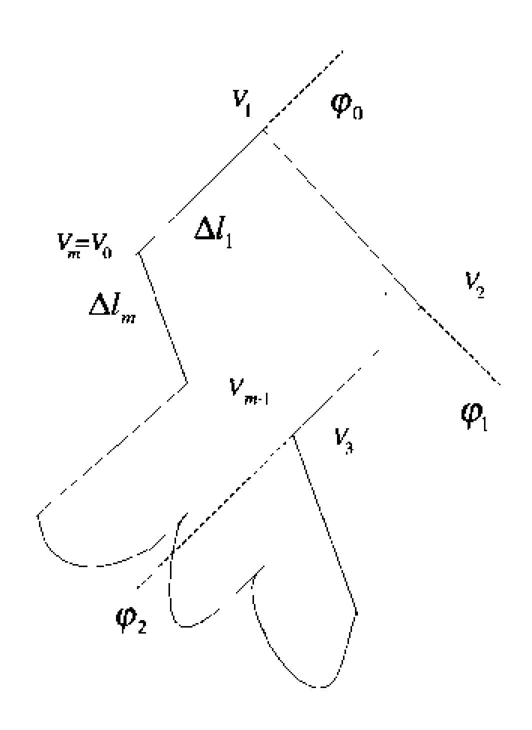


图 5-8　数字图像下的多边形边界

数字图像中，封闭曲线 r 通常是由折线构成的多边形或可用多边形来近似。设多边形顶点为 $V_0,V_1,\cdots,V_{m-1}$，边 $V_{i-1}V_i$ 的长度为 $\Delta l_i(i=1,2,\cdots,m)$，如图 5-8 所示。其中 Δl_m 是边 $V_{m-1}V_0$ 之长。在每边 $V_{i-1}V_i$ 上，$\varphi(l)$ 为常数，设为 φ_{i-1}，定义 $l_k=\sum_{i=1}^{k}\Delta l_i$，$\lambda=\frac{tL}{2\pi}$ 则：

$$a_0=\frac{1}{2\pi}\int_0^{2\pi}\varphi^*(t)\mathrm{d}t=\frac{1}{L}\int_0^{L}\varphi(\lambda)\mathrm{d}\lambda+\pi=-\pi-\frac{1}{L}\sum_{k=1}^{m}l_k(\varphi_k-\varphi_{k-1})$$

$$a_n=\frac{2}{L}\int_0^{L}\left[\varphi(\lambda)+\frac{2\pi\lambda}{L}\right]\cos\frac{2\pi\lambda n}{L}\mathrm{d}\lambda=\frac{2}{L}\sum_{k=0}^{m-1}\int_{l_k}^{l_{k+1}}\left[\varphi(\lambda)+\frac{2\pi\lambda}{L}\right]\cos\frac{2\pi\lambda n}{L}\mathrm{d}\lambda$$

$$=-\frac{1}{n\pi}\sum_{k=1}^{m}(\varphi_k-\varphi_{k-1})\sin\frac{2\pi nl_k}{L}$$

$$b_n=\frac{1}{n\pi}\sum_{k=1}^{m}(\varphi_k-\varphi_{k-1})\cos\frac{2\pi nl_k}{L} \tag{5-65}$$

其中 $n=1,2,\cdots$。这样，区域边界 r 就可用序列 $\{a_0,a_1,b_1,a_2,b_2,\cdots\}$ 进行描述和刻画。

（2）Person－Fu 傅立叶描绘子

上面求 a_n 和 b_n 时，已经指出对数字图像而言，$\varphi(l)$ 是分段连续的，即在（V_{k-1},V_k）边上是常数，而在端点上是不连续的，存在跳变。这会导致傅立叶变换中产生高频分量，因而在用 Zahn 描述时，常要较多的傅立叶级数，以保证信息不会有大的丢失。

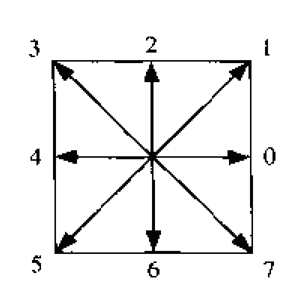
Person 和 Fu 将 r 与下面的复参量对应：$u(l)=x(l)+iy(l)$。其中 l 仍然是弧长，$x(l)$ 和 $y(l)$ 分别是曲线上点的横坐标及纵坐标。相应的傅立叶变换为：

$$u(l)=\sum_{-\infty}^{+\infty}a_n\exp(i2nl\pi/L) \tag{5-66}$$

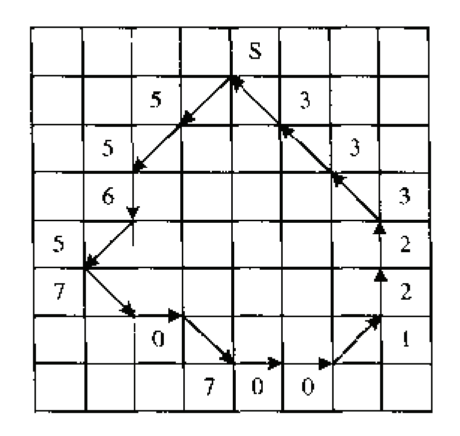
$$a_n=\frac{1}{L}\int_0^{L}u(l)\exp(-i2nl\pi/L)\mathrm{d}l=\frac{1}{L(2n\pi/L)^2}\sum_{k=1}^{m}(b_{k-1}-b_k)\exp(-i2nl_k\pi/L)$$

其中 $l_k=\sum_{i=1}^{k}\left|V_i-V_{i-1}\right|$，$b_k=\frac{V_{k+1}-V_k}{\left|V_{k+1}-V_k\right|}$，$V_k$ 是多边形第 k 个顶点的复数坐标。

经过同 Zahn 方法类似的归一化处理，$\{a_n\}$ 对 r 的平移、旋转和尺度变换具有不变性。适当地取 $\{a_n\}$ 的前几项就可在信息损失较小的前提下描述边界 r。



(a) 边界的 8 种走向和对应的方向码



(b) 边界产生的方向链码

图 5-9

2. 链码

对于离散的数字图像，区域的边界轮廓可理解为相邻边界像素点之间的单元连线逐段相连而成。考虑数字图像像素点(x,y)的一个 8 邻域，显然在该点处的边界只能在以下几个方向：正东、东北、正北、西北、正西、西南、正南和东南，如图 5-9(a)所示。对于每一种方向赋以一种码表示，如上面 8 个方向分别对应于 0、1、2、3、4、5、6 和 7 ，这些码称为方向码。

假设从某一个起点开始，将边界的走向按上面的编码方式记录下来，可形成如下的序列 $a_1\, a_2\, a_3 \cdots a_n$。$a_1 \sim a_n$ 取值为 0～7，这一序列称为链码的方向链。再加上一些标识码，即可构成链码。从图 5-9(a)可知，偶数链码段为垂直或水平方向的代码段，奇数链码段为对角线段。对图 5-9(b)所示的一个图像区域，若以 S 点为出发点，按逆时针的方向进行，所构成的边界链码应为 556570700122333。当然，也可以按顺时针方向进行，所构成的边界链码完全不同逆时针方向行进的情况。因此，边界链码具有行进的方向性，在具体应用时必须加以注意。

有了链码的方向链后，再加上一些标识码，即可构成链码。常用的标识码有两种：

（1）加上特殊专用的链码结束标志。如采用“!”作为结束标志，则图 5-9(b)的链码应为 556570700122333!。

（2）标上起始点的坐标。如图 5-9(b)的链码为 556570700122333XYZ，XYZ 为起始点 S 的坐标，用 3 位 8 进制数表示。

从链码可以得出边界的许多形状特征：

（1）链的长度

$$L = n_e + n_o \sqrt{2} \tag{5-67}$$

其中 n_e 表示方向链中偶数码的数目，n_o 表示奇数码的个数。奇数码用 $\sqrt{2}$ 的权重修正后，边界的长度检测时可以弥补图像离散化的误差。

（2）边界所表示区域的宽度和高度

设方向链为$\{a_1\, a_2\, a_3 \cdots a_n\}$，定义 a_i 在 X 轴上的分量为 a_{ix}，在 Y 轴上的分量为 a_{iy}，则：

a_i=0 时，a_{ix}=1，a_{iy}=0

a_i=1 时，a_{ix}=1，a_{iy}=1

a_i=2 时，a_{ix}=0，a_{iy}=1

a_i=3 时，$a_{ix}=-1$，$a_{iy}=1$

a_i=4 时，$a_{ix}=-1$，$a_{iy}=0$

a_i=5 时，$a_{ix}=-1$，$a_{iy}=-1$

a_i=6 时，$a_{ix}=0$，$a_{iy}=-1$

a_i=7 时，$a_{ix}=1$，$a_{iy}=-1$

设 x_0 和 y_0 是起始点的坐标，则：

$$
\begin{aligned}
\text{宽度} &= \max_i\left(\sum_{k=1}^{i} a_{kx} + x_0\right) - \min_i\left(\sum_{k=1}^{i} a_{kx} + x_0\right) \\
\text{高度} &= \max_i\left(\sum_{k=1}^{i} a_{ky} + y_0\right) - \min_i\left(\sum_{k=1}^{i} a_{ky} + y_0\right)
\end{aligned} \tag{5-68}
$$

（3）链码所包围的区域面积：

$$
\begin{aligned}
S &= \sum_{i=1}^{n} a_{ix}(y_{i-1} + a_{iy}/2) \\
y_i &= \sum_{k=1}^{i} a_{ky} + y_0
\end{aligned} \tag{5-69}
$$

（4）假如两个像素点可由方向链联接$\{a_1 a_2 a_3 \cdots a_n\}$，则这两点的距离为：

$$
d = \sqrt{\left(\sum_{i=1}^{n} a_{ix}\right)^2 + \left(\sum_{i=1}^{n} a_{iy}\right)^2} \tag{5-70}
$$

上述的区域特征用链码来计算，计算比较简单。但是在描述形状时，信息并不完全，这些数值特征与具体的形状之间并不一一对应。因此，不能只用这些数值进行形状识别，必须与其他特征信息相结合使用，作为补充信息，却能大大提高系统的识别性能。

5.4　颜色特征提取

有关颜色特征已经在彩色图像分割中有所描述，而作为图像识别，颜色依然是一个重要的特征。它是与纹理特征，形状特征完全不同的特征信息空间，与其他特征相比，颜色特征具有信息丰富的优点，但是图像的数据量急剧增加。有关颜色空间的选择，以及不同颜色空间所代表特征量已在第 4 章的图像分割有过论述。

第6章　图像识别

前面介绍了图像的变换和增强等技术，它们都是对输入图像的某种有效的改善，其输出仍然是一幅完整的图像。

随着数字图像处理技术的发展和实际应用的需求，出现了另一类问题，就是不要求其结果输出是一幅完整图像，而是将经过上述处理后的图像，再经过分割和描述提取有效的特征，进而加以判决分类。例如要从遥感图像中分割出各种农作物、森林资源和矿产资源等，并进一步判断其产量或蕴藏量，根据医学涂片分析发生病变的细胞形状和颜色判断是否发生癌变，交通管理系统中应用车牌自动识别技术管理车辆，AGV 自导引小车的路径自动识别以及机械加工零件质量的自动测量等。本章在介绍图像识别基本概念的基础上，重点介绍了统计模式识别方法、结构语句识别方法、模糊集识别方法和神经网络识别方法。

6.1　图像识别概述

图像识别（Pattern Recognition，也称模式识别，这里沿用习惯名称），简单地说就是要把一种研究对象，根据其某些特征进行识别并分类。例如要识别写在卡片上的数码字，判断它是0,1,2,…,9 中的哪个数字，就是将数码字图像分成十类的问题。因此，可以认为把图像进行区别分类就是图像的模式识别。这种识别早已存在于人们的生活实践中。然而，随着实践活动的扩大、深入和更加社会化的需要，人们不仅需要识别分类数很多的事物，而且被识别的对象的内容也越来越复杂。特别是由于科学技术水平的提高，使得各种不同的研究对象“图像化”或“数字化”，也就是说，可采用某种技术把考察的对象转换成照片（如各文档扫描）、波形图（心电图、地震波等）以及若干数据（遥测遥感中用多光谱扫描所得的数据），这些数据就可以代表所研究的对象。但是对模式识别来说，无论是数据、信号还是平面图像或立体景物都是除掉它们的物理内容而找出它们的共性，把具有同一共性的归为一类，而具有另一种共性者归为另一类。模式识别的目的就是研制采用某种仪器或设备，自动处理某些信息，代替人完成分类和辨识的任务，并且能够快速而准确地进行图像识别。

一个图像系统可以分为 4 个主要部分，其框图如图 6-1 所示。

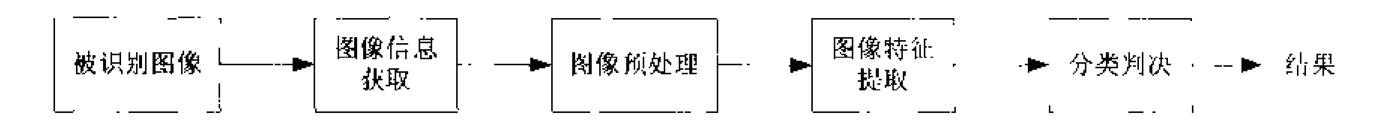


图 6-1　图像识别的简单框图

第 1 部分是图像信息的获取，它相当于对被研究对象的调查和了解，从中得到数据和材料。对图像识别来说，就是把图片、文字图形等用摄像头或光电扫描设备转换成电信号后输入计算

机以备后续处理。第2部分是图像预处理。预处理的目的是去除干扰、噪声及差异，将原始图像变成适合于计算机进行特征提取的形式。它包括图像的变换和增强等，这些内容在本书的前面章节已做过介绍。第3部分是图像特征提取。它的作用在于把调查了解到的数据材料进行加工、整理、分析和归纳，以去伪存真，去粗取精，抽出能反映事物本质的特征。当然，提取什么特征，保留多少特征与采用何种判决有很大关系。第4部分是分类判决。即根据提取的特征参数，采用某种分类判别函数和判别规则，对图像信息进行分类和辨识，得到识别的结果。这相当于人们从感性认识上升到理性认识而做出结论的过程。第4部分与特征抽取的方式密切相关，它的复杂程度也依赖于特征的抽取方式，例如，类似度、相关性和最小距离等。

6.2　统计模式的识别方法

统计模式识别的目的在于确定已知样本所属的类别。它以数学上的决策理论为依据，并根据此理论建立了统计学识别模型。其基本模型是在对研究的图像进行大量统计分析，找出规律性认识，抽出反映图像本质特点的特征进行识别。

统计模式的识别框图如图6-2所示。图中上半部分是识别部分，即对未知类别的图像进行分类；下半部分是分析部分，即对已知类别的图像样本制定出判决函数及判决规则（有规则的学习），使得对未知类别的图像能够进行分类。由于所输入的图像需要进行数字化，这就会产生误差；光照的不均匀，环境中存在的噪声干扰会损坏图像的质量等。所有这些都需要进行预处理。经过预处理的图像进行特征提取，最后进行判决分类，得到识别结果。为了进行分类，必须有图像样本。框图右下角部分是学习训练部分。当用训练图像样本根据某些准则制定（学习）出一些判决规则后，再对这些训练样本逐个进行检测，观察是否有误差（这相当于请老师进行指导），如果有的话，再进一步改进判决规则，直到比较满意为止。

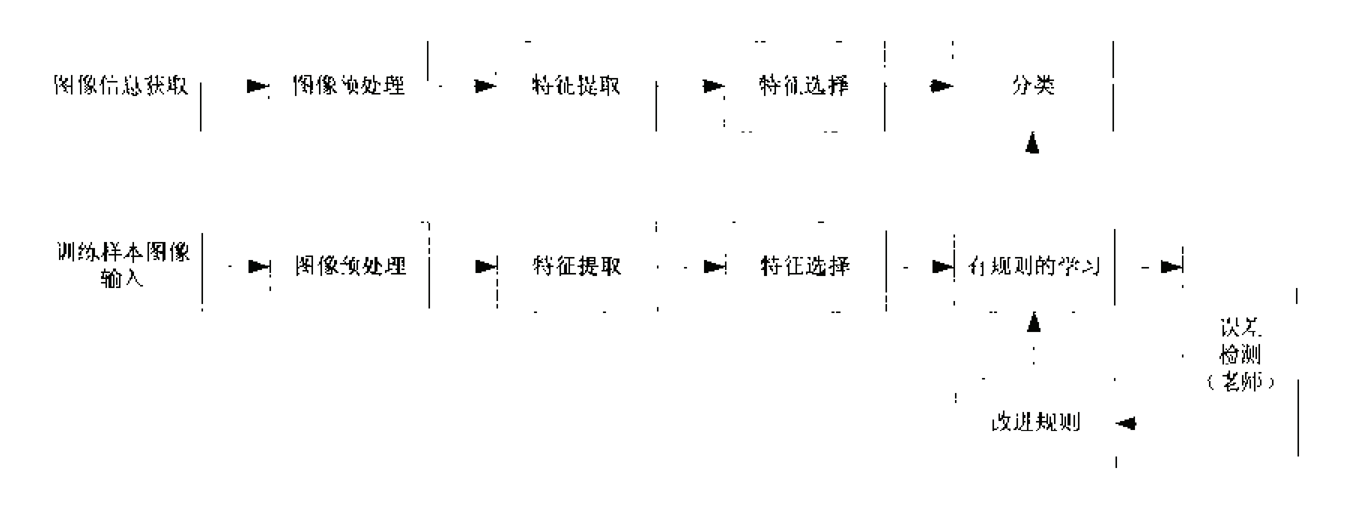


图6-2　统计模式识别框图

6.2.1　决策理论方法

正如图6-2的框图所示，统计模式识别方法最终归结为分类问题。假如已抽取出N个特征，而图像可分为m类，那么就可以对N进行分类，从而决定未知图像属于m类中的哪一类。一般把识别模式看成是N维空间中的向量X，即：

$$X = [x_1 \quad x_2 \quad \cdots \quad x_N]^{\mathrm{T}}$$

模式类别为 $w_1,w_2,\cdots,w_m$，识别就是要判断 X 是否属于 w_i 以及属于 w_i 中的哪一类。在这个过程中主要解决两个问题：一是如何抽取特征，要求特征数 N 尽可能小而且对分类判断有效，二是假设已有了代表模式的向量，如何决定它属于哪一类，这就需要判别函数。例如，模式有 $w_1,w_2,\cdots,w_m$ 共 m 个类别，则应有 $D_1(X),D_2(X),D_3(X),\cdots,D_m(X)$ 共 m 个判别函数。如果 X 属于第 i 类，则有：

$$D_i(X) > D_j(X) \qquad (j = 1,2,\cdots,m;\ j \neq i)$$

在两类的分界线上，则有 $D_i(X)=D_j(X)$

这时 X 既属于第 i 类，也属于第 j 类，因此这种判别失效。为了进行识别就必须重新考虑其他特征，再进行识别。问题的关键是找到合适的判别函数。

1. 线性判别函数

线性判别函数是一种应用较广的一种判别函数。所谓线性判别函数，是指判别函数是图像所有特征向量的线性组合，即：

$$D_i(X) = \sum_{k=1}^{N} w_{ik} X_k + w_{i0} \qquad i = 1,2,\cdots m \tag{6-1}$$

式中 $D_i(X)$代表第 i 个判别函数，w_{ik} 是系数或权重，w_{i0} 为常数或称为阈值。在两类之间的判决界处有：

$$D_i(X) - D_j(X) = 0 \tag{6-2}$$

该方程在二维空间是直线，在三维空间是平面，在 N 维空间则是超平面。$D_i(X)-D_j(X)$ 可以写成以下的形式：

$$D_i(X) - D_j(X) = \sum_{k=1}^{N} (w_{ik} - w_{jk}) X_k + (w_{i0} - w_{j0}) \tag{6-3}$$

其判决过程可如下进行：如果 $D_i(X)>D_j(X)$，或 $D_i(X)-D_j(X)>0$，则 $X\in w_i$，如果 $D_i(X)< D_j(X)$ 或 $D_i(X)-D_j(X)<0$，则 $X\in w_j$。

用线性判别函数进行分类是线性分类器。任何 m 类问题都可以分解为（m-1）个二类识别问题。方法是先把模式空间分为一类和其他类，如此进行下去即可。因此，两类线性分类器是最简单和最基本的。

分离两类的判决界由 $D_1-D_0=0$ 表示。对于任何特定的输入模式必须判定 D_1 大还是 D_2 大。若考虑某个函数 $D=D_1-D_2$，对于 1 类模式 D 为正，对于二类模式 D 为负。于是，只要处理与 D 相应的一组权输入模式并判断输出符号即可进行分类。执行这种运算的分类器的原理框图如图 6-3 所示，其中 Σ 是累加器。

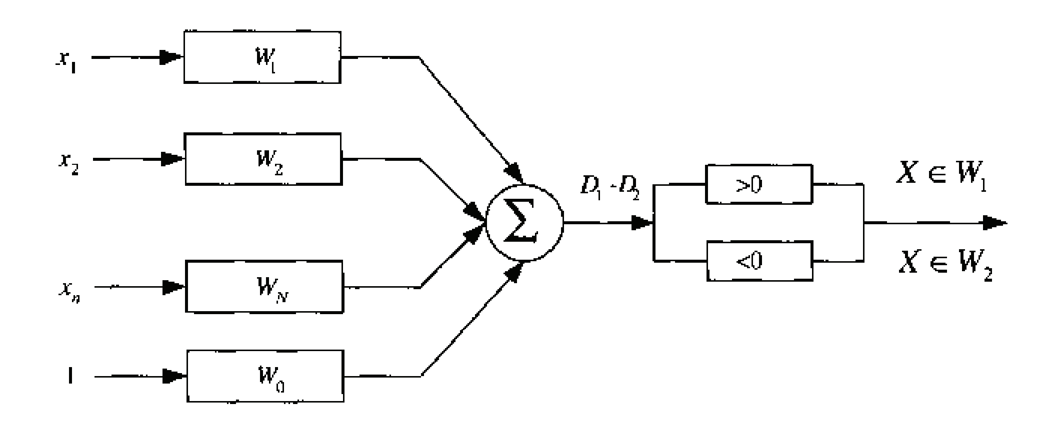


图 6-3　两类线性分类器

在线性分类器中要找到合适的系数，以便使分类尽可能不出差错，惟一的办法就是试验法。例如，先设所有的系数为 1，送进每一个模式，如果分类有错就调整系数，这个过程就叫做线性分类器的训练或学习。例如，把 N 个特征 X 和 1 放在一起叫做 Y，N+1 个系数为 W，即：

$$Y = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_N \\ 1 \end{bmatrix} \quad W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \\ w_{N+1} \end{bmatrix} \tag{6-4}$$

则线性判别函数可以改写为：

$$D(X) = Y^{\mathrm{T}}W \tag{6-5}$$

考虑分别属于两个不同模式类，m=2，此时有两个训练集 T_1 和 T_2。两个训练集合是线性可分的，这意味着存在一个加权向量 W：

$$\begin{cases} Y^{\mathrm{T}}W > 0 & Y \in T_1 \\ Y^{\mathrm{T}}W < 0 & Y \in T_2 \end{cases} \tag{6-6}$$

式中 Y^{T} 是 Y 的转置。如果分类器的输出不能满足式（6-6）的条件，可以通过“误差校正”的训练步骤对系数加以调整。例如，如果第一类模式 $Y^{\mathrm{T}}W$ 不大于零，则说明系数不够大，可用加大系数的方法进行误差修正。具体修正方法如下：

对于任一 $Y \in T_1$，若 $Y^{\mathrm{T}}W \leqslant 0$，则使：

$$W' = W + \alpha Y \tag{6-7}$$

对于任一 $Y \in T_2$，若 $Y^{\mathrm{T}}W > 0$，则使：

$$W' = W - \alpha Y \tag{6-8}$$

通常使用的误差修正方法有固定增量规则，绝对修正规则及部分修正规则。

（1）固定增量规则：选择 α 为一个固定的非负数。

（2）绝对修正规则：取 α 为一最小整数，它可使 $Y^{\mathrm{T}}W$ 的值刚好大于零。即：

$$\alpha = 大于\frac{\left|Y^{\mathrm{T}}W\right|}{Y^{\mathrm{T}}Y}的最小整数 \tag{6-9}$$

（3）部分修正规则：可取α为下式所决定的值：

$$\alpha=\beta\frac{\left|Y^{\mathrm{T}}W\right|}{Y^{\mathrm{T}}Y} \quad 0<\beta\leqslant 2 \tag{6-10}$$

2. 最小距离分类器

图像识别中，线性分类器中重要的一类是用输入模式与特征空间作为模板的点（标准样本的中心），它们之间的距离作为分类的准则。对于 m 类模板，未知类别图像与其中某一类的距离最近就属于该类。

假定图像有 m 类，分别为 $W_1,W_2,\cdots,W_m$，并有 m 个参考向量 $R_1,R_2,R_3,\cdots,R_m$，R_i与模式类W_i相联系。对于 R_i的最小距离分类就是把输入的新模式 X 分为 W_i类，其分类准则就是 X 与参考模型原型 $R_1,R_2,R_3,\cdots,R_m$之间的距离，跟哪一个最近就属于哪一类。X 和 R_i之间的距离可表示为：

$$\left|X-R_i\right|=\sqrt{(X-R_i)^{\mathrm{T}}(X-R_i)} \tag{6-11}$$

其中$(X-R_i)^{\mathrm{T}}$是$(X-R_i)$的转置，由式（6-11）可得：

$$\begin{aligned}\left|X-R_i\right|^2&=(X-R_i)^{\mathrm{T}}(X-R_i)\\&=X^{\mathrm{T}}X-X^{\mathrm{T}}R_i-R_i^{\mathrm{T}}X+R_i^{\mathrm{T}}R_i\\&=X^{\mathrm{T}}X-(X^{\mathrm{T}}R_i+R_i^{\mathrm{T}}X-R_i^{\mathrm{T}}R_i)\end{aligned} \tag{6-12}$$

在式（6-12）中，$X^{\mathrm{T}}X$ 与 R_i无关，由此可设定最小距离判别函数 $D(X)$为：

$$D_i(X)=X^{\mathrm{T}}R_i+R_i^{\mathrm{T}}X-R_i^{\mathrm{T}}R_i \tag{6-13}$$

最小化距离$\left|X-R_i\right|^2$，也就是最大化 $D_i(X)$。由上面的判别函数，在分类中，如果$X\in W_i$，则 $D_i(X)$=max。由式（6-13）可见，$D_i(X)$是一个线性函数，因此最小距离分类器也是一个线性分类器。在最小距离分类中，在决策边界上的点与相邻两类都是等距离的，这种方法就难于解决，此时，必须寻找新的特征，重新分类。

这种分类还可以用决策区域来表示。例如有二类问题 W_1，W_2，其模板分别为 R_1，R_2，当距离 $d(X,R_1)<d(X,R_2)$，或者：

$$\left[\sum_{i=1}^{n}(X_i-R_1)^2\right]^{\frac{1}{2}}<\left[\sum_{i=1}^{n}(X_i-R_2)^2\right]^{\frac{1}{2}}$$

则 $X\in W_1$，并可用决策区域来表示，如图 6-4 所示。



图 6-4　两类问题决策区域

将模板 R_1、R_2 作连线，再作平分线，平分线左边为 R_1 区域，平分线右边为 R_2 区域，R_1R_2 为决策区域，中间为决策面。在这种分类中，两类情况界面为线，决策区为两平面。对于三类情况，界面为超平面，决策区为半空间。

3. 最近邻域分类法

最近邻域分类法是图像识别中应用较多的一种方法。在最小距离分类法中，是取一个最标准的向量作为代表。将这类问题稍微扩张一下，一类不能只取一个代表，把最小距离的概念从一个点和一个点之间的距离扩充到一个点和一组点之间的距离。这就是最近邻域分类法的基本思路。设 $R_1,R_2,R_3,\cdots,R_m$ 分别是与类 $W_1,W_2,\cdots,W_m$ 相对应的参考向量的 m 个集合，在 R_i 中的向量为 R_i^k，即 $R_i^k \in R_i$，$k=1,2,\cdots,l_i$，也就是：

$$R_i = \{R_i^1, R_i^2, \cdots, R_i^{l_i}\}$$

输入特征向量 X 与 R_i 之间的距离用下式表示：

$$d = (X, R_i) = \min_k \left| X - R_i^k \right| \tag{6-14}$$

这就是说，X 和 R_i 之间的距离是 X 和 R_i 中每一个向量的距离中的最小者。空间点之间距离的求取方法与最小距离与最小距离分类方法中的距离求取法相同。

采用这种判别函数的图像识别，决策边界将是分段线性的。例如，如图 6-5 所示，有一个两类判别问题，W_1 类的代表为 R_1^1，R_1^2，W_2 类的代表为 R_1^1、R_1^2。如果有一个模式送入识别系统，首先要计算它与每个点的距离，然后找最短距离。这种方法的概念简单，分段线性边界可以代表很复杂的曲线，也可能本来是非线性边界，现在可用分段线性来近似代替。

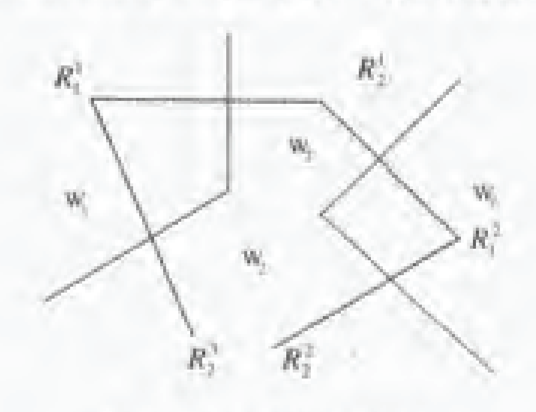


图 6-5　两类最近邻域分类

4. 非线性判别函数

线性判别函数很简单，但也有缺点。它对于较复杂的分类往往不能胜任。在较复杂的分类问题中就要提高判别函数的次数，因此根据问题的复杂性，可将判别函数从线性推广到非线性。非线性判别函数可写成下式的形式

$$D(x) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_Nx_N +$$

$$w_{12}x_1x_2 + w_{13}x_1x_3 + \cdots + w_{1N}x_1x_N +$$

$$w_{11}x_1^2 + w_{22}x_1^2 + \cdots + w_{NN}x_N^2$$

$$= w_0 + \sum_{k=1}^{N} w_{kk}x_k^2 + \sum_{k=1}^{N} w_kx_k + \sum_{k=1}^{N}\sum_{i=1}^{N} w_{ki}x_kx_i \qquad (6\text{-}15)$$

上式是一个二次型判别函数，通常二次型判别函数的决策边界是一个超二次曲面。

6.2.2 统计分类法

以上谈到的分类方法是在没有噪声干扰的情况下进行的，此时测得的特征的确能代表模式。如果在抽取特征时有噪声，那么可能抽取的特征代表不了模式，这时就要用统计分类法。用统计方法对图像进行特征抽取、学习和分类是研究图像识别的主要方法之一，而统计方法的最基本内容之一是贝叶斯分析，其中包括贝叶斯决策方法、分类器、估计理论、贝叶斯学习、贝叶斯距离等等。

1. 贝叶斯法则

在古典概率中贝叶斯定理已为大家所熟习：

$$P(B_i / A) = \frac{P(B_i)P(A/B_i)}{\sum_{j=1}^{n} P(B_j)P(A/B_j)} \qquad (6\text{-}16)$$

式中 $B_1,B_2,\cdots,B_n$ 是 n 个互不相容的事件，$P(B_i)$是事件 B_i 的先验概率，$P(A/B_i)$是 A 在 B_i 已发生条件下的条件概率。贝叶斯定理说明在给定了随机事件 $B_1,B_2,\cdots,B_n$ 的各先验概率 $P(B_i)$，以及条件概率 $P(A/B_i)$时，可计算出当事件 A 出现时，确定事件 B_i 出现的后验概率为 $P(B_i/A)$。

贝叶斯公式常用于分类问题中。假如设 X 表示事件的状态或特征的随机变量，它可以表示图像的灰度或形状等，设 w_i 表示事件类别的离散随机变量。根据特征 X 对图像进行分类，就可以用如下的公式：

$$P(w_i / X) = \frac{P(X / w_i)P(w_i)}{\sum_{i=1}^{m} P(w_i)P(X / w_i)} \qquad (6\text{-}17)$$

式中 $P(w_i)$称为 w_i 的先验概率，它表示事件属于 w_i 的预先粗略了解，$P(X/w_i)$表示事件属于 w_i 类而具有 X 状态的条件概率，$P(w_i/X)$叫做 X 条件下 w_i 的后验概率，它表示对事件 X 的状态作观察后判断属于 w_i 类的可能性。由式（6-17）可见，只要类别的先验概率及 X 的条件概率为

已知，就可以得到类别的后验概率，再加上最小误差概率或最小风险法则，就可以进行统计判决分类。

2. 最小错误率的贝叶斯分类法

假设有两个类别，每类有两种统计参数代表，即：

w_1：$P(w_1)$，$P(X/w_1)$

w_2：$P(w_2)$，$P(X/w_2)$

其中 $P(w_1)$，$P(w_2)$是先验概率，$P(X/w_1)$、$P(X/w_2)$是条件概率密度函数。在噪声干扰的影响下，每个模式只能得到某一类模式的概率分布。

利用贝叶斯法则分类：

若 $P(w_1)P(X/w_1) > P(w_2)P(X/w_2)$，则分类 $X \in w_1$。

反之 $P(w_2)P(X/w_2) > P(w_1)P(X/w_1)$，则分类 $X \in w_2$。

在上述分类法则中，乘积 $P(w_i)P(X/w_i)$ 起到了判别函数的作用。为了方便起见，在应用中，通常取乘积的对数形式：

$$\left.\begin{aligned}\lg P(w_1)P(X/w_1) > \lg P(w_2)P(X/w_2), \quad X \in w_1 \\ \lg P(w_2)P(X/w_2) > \lg P(w_1)P(X/w_1), \quad X \in w_2\end{aligned}\right\} \tag{6-18}$$

或者：

$$\left.\begin{aligned}\lg\frac{P(X/w_1)}{P(X/w_2)} > \lg\frac{P(w_2)}{P(w_1)} \qquad X \in w_1 \\ \lg\frac{P(X/w_1)}{P(X/w_2)} < \lg\frac{P(w_2)}{P(w_1)} \qquad X \in w_2\end{aligned}\right\} \tag{6-19}$$

对于两类的分类问题，分界面为：

$$\lg P(w_1)P(X/w_1) - \lg P(w_2)P(X/w_2) = 0 \tag{6-20}$$

在工程上很多问题，如产品的合格率等，是服从正态分布。设其均值为 M_i，协方差矩阵是 K_i，设 m=2，若 $P(X/w_i)$是正态分布，即：

$$P(x/w_i) \sim N(M_i, K_i)$$

可得到其决策分界面如下：

$$P(X/w_i) = (2\pi)^{-\frac{N}{2}}\left|K_i\right|^{-\frac{1}{2}}\exp\left[-\frac{1}{2}(X-M_i)^{\mathrm{T}}K^{-1}(X-M_i)\right] \tag{6-21}$$

当 i=1,2 时，如果 $X \in w_1$，由式（6-19）和式（6-21）得：

$$-\frac{1}{2}\lg\left|\frac{K_1}{K_2}\right| - \frac{1}{2}\left[(X-M_1)^{\mathrm{T}}K_1^{-1}(X-M_1)\right]$$

$$+\frac{1}{2}\left[(X-M_2)^{\mathrm{T}}K^{-1}(X-M_2)\right]>\lg\frac{P(w_2)}{P(w_1)} \tag{6-22}$$

此时两类间的决策边界是二次的。

若两个协方差矩阵相同，即 $K_1=K_2=K$，则

$$\left.\begin{aligned}X^{T}K^{-1}(M_1-M_2)+\frac{1}{2}(M_1+M_2)^{T}K^{-1}(M_1-M_2)>\lg\frac{P(w_2)}{P(w_1)} \quad 则X\in w_1\\ X^{T}K^{-1}(M_1-M_2)+\frac{1}{2}(M_1+M_2)^{T}K^{-1}(M_1-M_2)<\lg\frac{P(w_2)}{P(w_1)} \quad 则X\in w_2\end{aligned}\right\} \tag{6-23}$$

边界决策成为线性。所以求两类的分类问题，若每类都是正态分布，但协方差矩阵不同，则分界就是二次函数；若 N 很大，求 K^{-1} 很麻烦，仍旧假设 $K_1=K_2$，那将会出错，例如在图像形状识别中，就可能把球形滚珠的问题变为平面的问题。

【例 6-1】假设在某条生产线上加工零件表面质量的识别中，合格(w_1)和不合格(w_2)两类的先验概率分别为：合格，$P(w_1)=0.9$；不合格，$P(w_2)=0.1$。

现有一待识别的零件，其表面图像特征的观察值为 x，从条件概率密度分布曲线上差得：

$$P(x/w_1)=0.2,\quad P(x/w_2)=0.4$$

试对该零件进行分类。

解：利用贝叶斯公式，分别计算出 w_1 和 w_2 的后验概率

$$P(w_1/x)=\frac{P(x/w_1)P(w_1)}{\sum_{i=1}^{2}P(x/w_i)P(w_i)}=\frac{0.2\times0.9}{0.2\times0.9+0.4\times0.1}=0.818$$

$$P(w_2/x)=1-P(w_1/x)=0.182$$

根据贝叶斯决策，有：

$$P(w_1/x)>P(w_2/x)$$

所以最终的识别结果为合格零件。

3. 最小风险的贝叶斯分类法

设 $X=\{x_1,x_2,\cdots,x_N\}$ 是随机变量，并且共有 m 个类，即 $w_1,w_2,\cdots,w_m$，对于每一类模式 w_i，其 $P(X/w_i)$及 $P(w_i)$都是已知的。以 $P(X/w_i)$及 $P(w_i)$为基础，一个分类器的成功条件是要在错判概率最小的条件下来完成分类任务。

定义一个决策函数 $a(X)$，其中 $a(X)=a_i$，表示假设 $X\in w_i$ 被接受。损失引入损失函数为 $L(a_j,w_i)$ 表示如果输入模式 X 实际是来自 w_i，而作出的决策是 a_j，即将它分类到 w_j 所产生的损失。

由于引入了“损失”的概念，在考虑错判所造成的损失时，就不能只根据后验概率的大小来进行分类，而必须考虑分类的结果是否使损失最小。在决策 a_i 情况下的条件期望损失为

$$R(a_i/X)=\sum_{j=1}^{m}L(a_i,w_j)P(w_j/X) \tag{6-24}$$

$R(a_i/X)$也称为条件风险。定义期望风险 R 为：

$$R = \int R(a(X)/x)P(X)\mathrm{d}X \tag{6-25}$$

式中，$\mathrm{d}X$ 是特征空间中的体积元，积分在整个特征空间进行。

选择适当的决策 $a(X)=a_i$，$i=1,2,\cdots,m$，以使期望风险 R 取最小值。如果在采取每一个决策时，都使条件风险最小，则对所有的 X 作出决策时，其期望风险也必然最小。这种决策称为最小风险贝叶斯决策。即如果$R(a_k/X)=\min\limits_i R(a_i/X)$，则 $a=a_k$，即 $X\in w_k$。

【例 6-2】在例 6-1 条件的基础上，按最小风险贝叶斯决策进行分类。已知 $L(a_1,w_1)=0$，$L(a_2,w_1)=1$，$L(a_1,w_2)=6$，$L(a_2,w_2)=0$。

解：已知条件为

$P(w_1)=0.9$　　　　$P(w_2)=0.1$

$P(X/w_1)=0.2$　　　　$P(X/w_2)=0.4$

$L(a_1,w_1)=0$　　　　$L(a_1,w_2)=6$

$L(a_2,w_1)=1$　　　　$L(a_1,w_1)=0$

参考例 6-1 的计算结果，可知：

$P(w_1/X)=0.818$　　　　$P(w_2/X)=0.182$

按式（6-24）计算出条件风险：

$$R(a_1/X)=\sum_{j=1}^{2}L(a_1,w_j)P(w_j/X)=L(a_1,w_2)P(w_2/X)=1.092$$

$$R(a_2/X)=\sum_{j=1}^{2}L(a_2,w_j)P(w_j/X)=L(a_2,w_1)P(w_1/X)=0.818$$

由于$R(a_1/X)>R(a_2/X)$，即决策为不合格零件。本例识别结果与例 6-1 恰好相反，这是因为影响识别结果的因素又多了一个，即“损失”。

6.3　结构语句的识别方法

6.3.1　概述

统计模式的识别方法是图像识别中应用较为广泛的一种方法。但这种方法对结构复杂、形式多变的图像特别是图像结构信息起着非常重要的作用，且识别的目的不仅在于把图像简单地分配到某一特定的类别中去，而且还要描述图像的性质，以便根据这些性质决定其属性时，统计方法就有很大的局限性。例如手写体文字及景物等的识别就是如此。对于这些图像，由于它们十分复杂，需要很多特征才能描述它们，但相同图像内容的多变形式又使得精确描述非常困难。人们通过大量实践，设计出了这样一种方法，即把一个复杂模式分化为若干较简单的子模式的组合，而子模式又分为若干基元、通过对基元的识别，进而识别子模式，最终识别该复杂模式。例如汉字、指纹、连续语音（不是一个个孤立的字母）的识别，往往都采用这一方法，并已取得可喜的成果。如图 6-6 所示的景物可用图 6-7 所示的层次结构来描述。

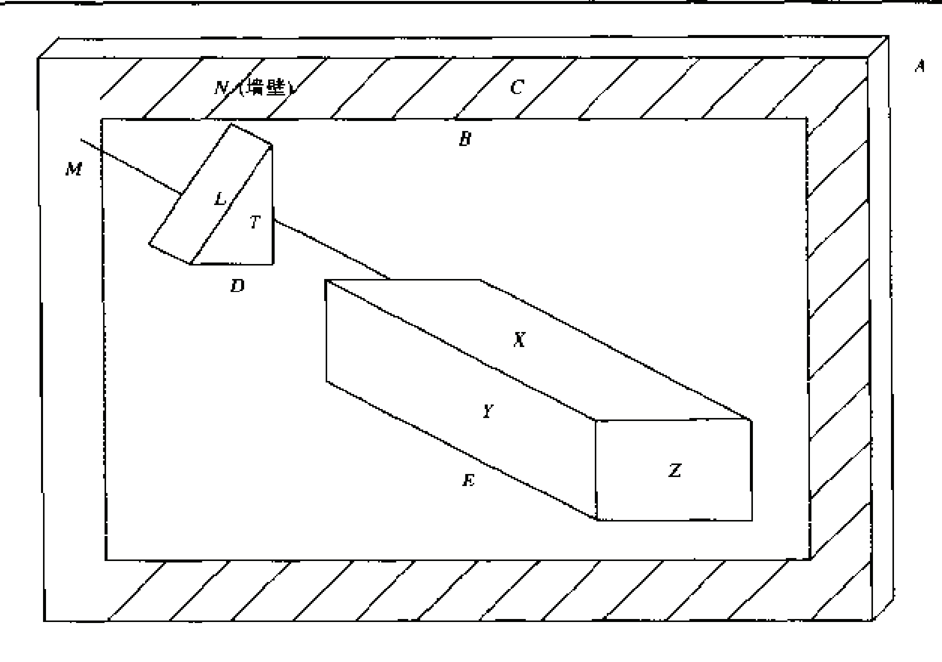


图 6-6　一个景物 A

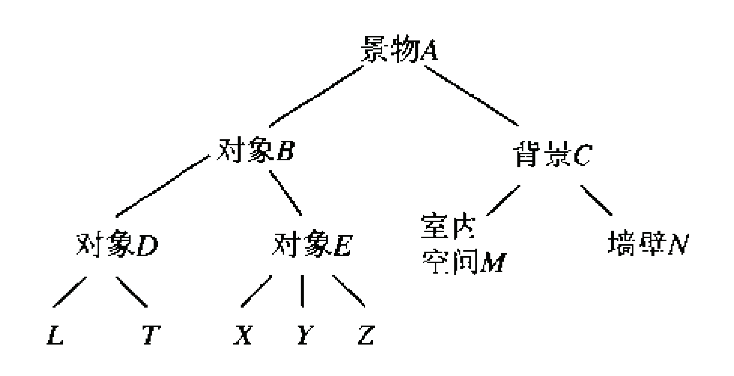


图 6-7　景物 A 及其多层次结构描述

为了表示每一层模式的层次结构，可以把模式描述的结构法类比于语言的语法。由基元、子模式以不同方式构成模式的过程，恰似由字构成词，由词构成句子的过程。显然，要使此种方法有优越性，就应选取远比模式本身更易于识别的最简单的子模式，即“模式基元”。用作模式的结构描述的语言包括两部分：即模式基元和对基元的合成操作规则，这种语言就被称为“模式描述语言”。对基元作合成操作以构成模式的规则，就叫做语法。当模式中的每一基元被辨认以后，识别过程就可通过执行语法分析来实现，亦即对描述待识模式的“句子”进行分析，看它是否遵守指定的语法。与此同时，语法分析还产生出该句子的结构描述，通常是树状的结构描述。

结构模式识别的方法，用小而简单的基元与语法规则来描述大而复杂的模式的能力。分析表明，这种方法还有一个特点，这就是利用了语法的递归性。因为同一语法规则，可以递归地应用任意多次，所以它就能以十分紧凑的形式，表示一个甚大的句子集合。当然这样一种方法的实用程度和基元的辨识能力有关，还和用合成操作表达基元间相互关系的能力有关。

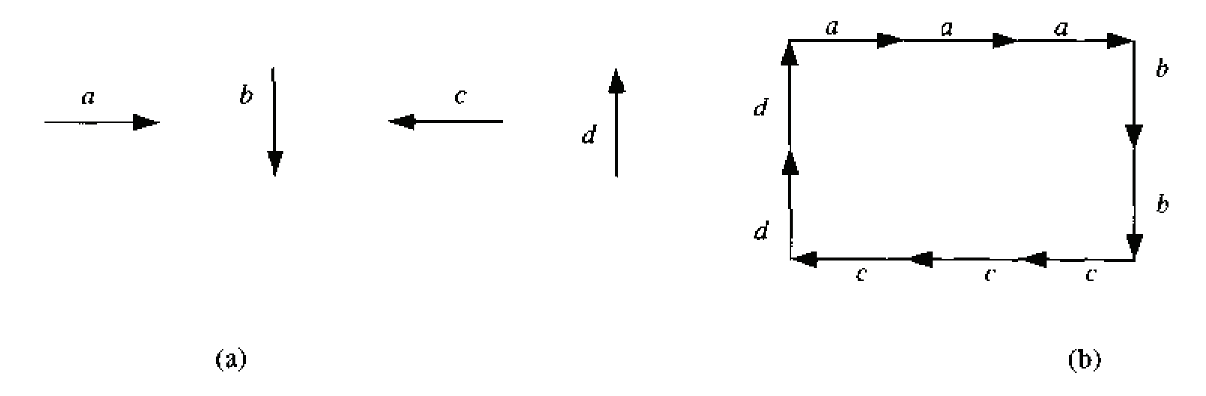


图 6-8　一个矩形及其模式基元

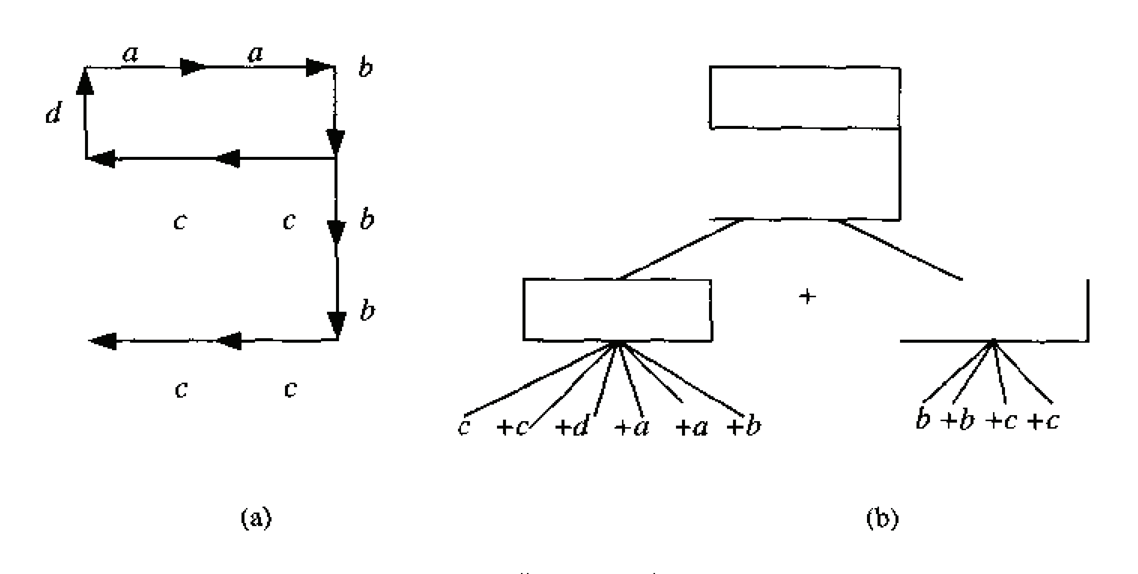


图 6-9　模式及其结构描述

通常可用“与”、“或”这样的逻辑操作算符来对子模式作合成操作。例如，仅选择“链接”为描述模式的惟一关系，且选取图 6-8(a)中所示的基元，则图 6-8(b)所示的方框可表示为串：*aaabbccccdd*。

进而，若用“+”表示“从头到尾的链接”这样一种操作，则图 6-8(b)的矩形可表示为 *a+a+a+b+b+c+c+c+d+d*。仍采用图 6-8(a)中的基元来表示图 6-9(a)所示的模式，则如图 6-9(b)所示。可以看出，在这种描述方法下，基元与基元之间的关系均被表示为串中的符号。

一种模式的结构信息，也可以用“关系图”来表示，如图 6-10 所示是景物图 6-6 所示的关系图。由于在关系图与矩阵间存在一一对应关系，一个关系图就无疑可表示为一个“关系矩阵”。在模式描述中采用关系图，就能扩充允许的关系的种类，以包含那些从模式中很容易确定的关系。

注意：①链接是一维语言惟一的基本操作；②一个图可以包含闭环，而树不允许有闭环。

用这种扩展了的形式，就能拥有比树状结构更强的描述手段。然而采用树状结构，却提供了一种直接的方法，使得正规的语言理论适应紧凑地描述问题的要求，并适于分析含有结构内容的模式。

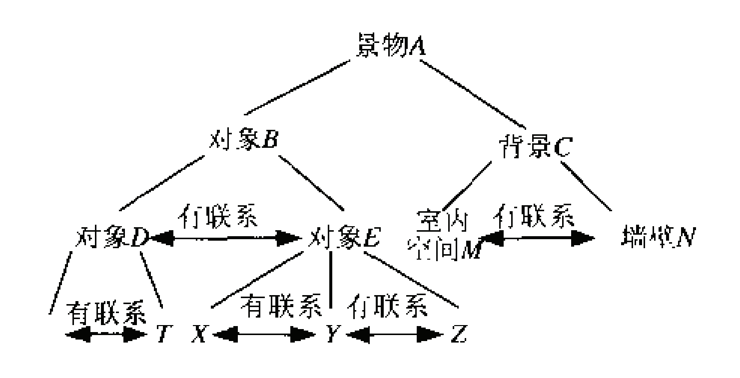


图 6-10 图像结构的关系图

6.3.2 结构模式识别系统

结构句法图像识别系统的简单框图如图 6-11 所示。它由识别及分析两部分组成。识别部分主要组成是：预处理、基元抽取（包括基元和子图像之间的关系）和结构分析。分析部分包括基元选择及结构推断。

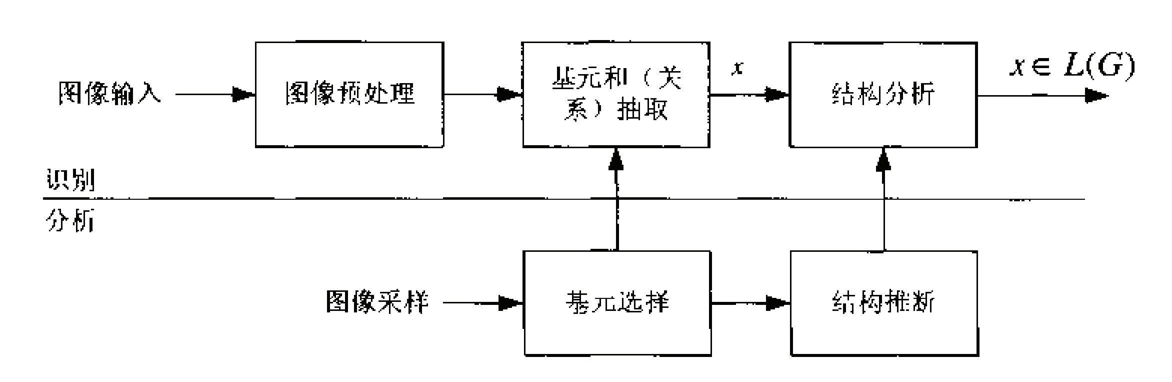


图 6-11 图像句法结构识别系统框图

每一个经过预处理的图像，在基元抽取阶段分割图像，同时抽取基元，并显示其相互关系，以便用它的子图像来进行描述。这就是预定的句法和组合运算，把图像分割为子图像及图像基元，并且各子图像本身又由给定的一组图像基元来识别。这样各个图像就由一组图像基元，按指定的句法运算，用 x 表达式表示出来了。

结构分析是指“结构分析器”，它是判别所得的表达式 x，在句法上是否正确（符合由给定文法或句法描述的图像类别）。如果句法上是正确的，则分析器进行句法分析，就能得到一个图像的完整描述。否则，该图像或者被拒识，或者就要用其他的文法来完成结构分析，并判别为其他类别的图像。

图像分析是为图像识别服务的，其中基元的选择是提供参考图像基元或原型图像基元，它也可以是参考图像的句子，供识别部分“模板匹配”用，使代表输入图像的基元链（或描述输入图像的句子）与代表各参考图像（或原型图像的基元链或句子）相匹配，确定出语句。然后再根据选定的“匹配”或“相似性”准则，根据输入图像能与哪个样板图像最好地匹配，则输入地图像就被分入那个样板的类别中去，这就是分析部分根据基元选择作结构推断，提供给识别部分作句法分析，确定图像类型，以完成图像识别任务。

6.3.3　图像基元的选择与抽取

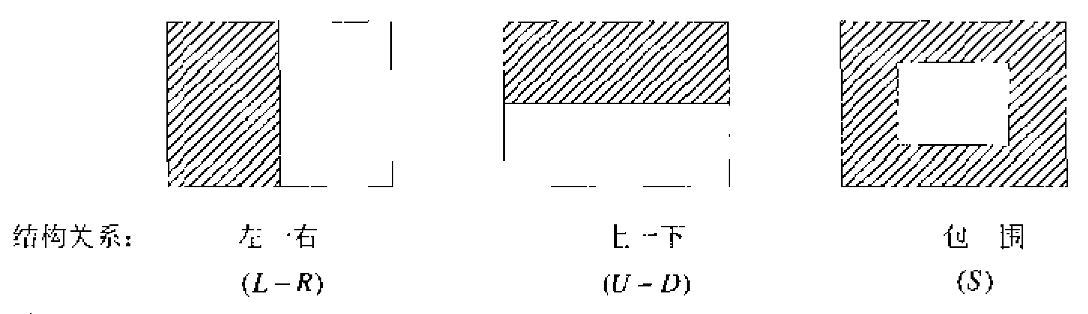
1．图像基元的选择

在句法结构识别系统中，首要的一步是要确定一个图像基元集，但这受制于图像本身的特征、用途、技术可行性等因素。目前，还没有统一的办法来解决基元选择问题，这里只能提出一些方向性的建议：①所选取的基元应是精简的，以易于语法描述与分析；②能用非语言学的方法方便地抽取，因为它们被认为是最基本的单元。

建议①与建议②有时是有矛盾的，这是因为根据目前的技术水平，按建议①来选择基元会使这些基元太复杂而难以识别。而另一方面建议②强调了基元的易于抽取，而可能会使描述与分析变得很复杂。这两者之间的折衷，有时对于识别系统的实现是很重要的。例如对数学表达式作结构分析时，用字符与运算符作基元就较简单，而直接选用直线或曲线线段作基元就很困难，亦即在这以前应该先做一步从笔划到字符（算符）的合成操作。

以汉字识别为例。根据汉字结构的研究，发现可用为数不多的分割操作。例如：

分割操作：



每种分割操作，在相邻的两子模式或基元间产生特定的结构关系。递归地应用这些操作，我们能将汉字分割成它的子模式与基元。若用目前的技术能抽取适当的基元的话，一个汉字可用给定的一组基元与结构关系来描述，图 6-12 就是对于汉字“桐”的例子。从这例子也可以看到，若用笔划作为基元，就会使结构描述比现在复杂得多。

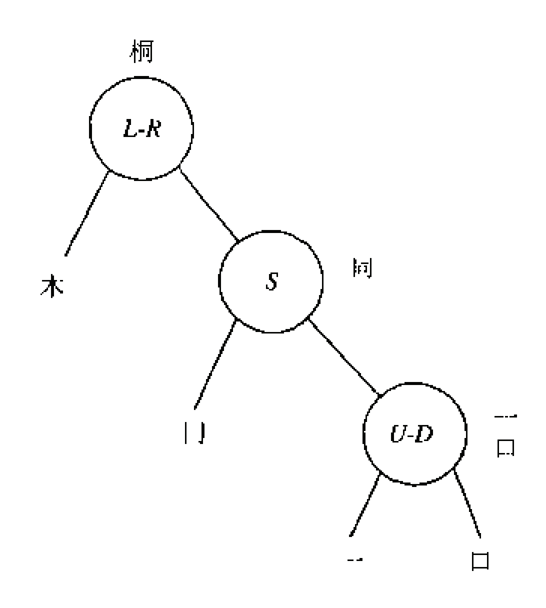


图 6-12　关于汉字“桐”的结构描述

2. 图像基元的抽取

正如前面讨论所指出的，没有一成不变的办法解决基元的选择问题，因而基元抽取也只能是在基元选定了以后，根据基元的特点才可能设计出相应的办法。本节将在第 8 章的数字字符图像的句法结构识别中详细介绍有关数字基本基元的提取。

6.3.4 图像描述语言、图像描述文法

用一组图像基元及其组合运算来表达图像结构的“语言”，常称为“图像描述语言”。图像由基元组成，基元组成图像时，必须遵循一定的规则，这种规则就用“文法”来描述，称为“图像描述文法”，亦称句法。图像识别时，首先选择图像中的基元，然后就是要构成一个文法，这种文法产生一种语言（或多种语言）构成的句子，以描述被研究的图像。再对图像的“句子”进行文法分析。文法分析对描述给定图像的句子产生一个结构描述，即树状结构图，以确定该句子对指定的文法在句法上是否正确。若正确就得到了一个确定语言，然后就能识别出该图像，否则查出错误，重新确定其语言，重新识别。

利用图像识别的文法方法，可用一组为数不多的简单图像基元和文法规则，来描述大量复杂的图像。在子图像中定义的各种关系或组合运算通常能以逻辑的和/或数学运算来表达，其特点是，一个文法规则可使用任意多次，它能以非常紧凑的方法表达句子的各种基本结构。另一特点，就是能利用文法上的归纳性质来进行分析。由此可见，一个图像能否被准确地识别，取决于是否能很好识别简单的图像基元，以及图像基元之间由组合运算所表示的相互关系。下面介绍常用的两类文法。

1. 短语结构文法

在下述形式的文法中，产生的一般形式为$\alpha \to \beta$，称为短语结构文法。它是一个四元式，表示为：

$$G = (V_N, V_T, P, S) \tag{6-26}$$

式中，V_N是非终止符有限集合，一般用大写英文字母表示：

$$V_N = \{A, B, \cdots\} \tag{6-27}$$

V_T是终止符的有限集合，用小写英文字母表示：

$$V_T = \{a, b, \cdots\} \tag{6-28}$$

对任何字母表 V 来说，在 V 上所有句子的可数无穷集合，包括空串λ是 V 的闭包，记作V^*。V 的正闭包是集合：

$$V^+ = V^* - \{\lambda\} \tag{6-29}$$

例如，对于给定的字母表 $V=\{a,b\}$，它的闭包和正闭包是：

$$V^* = \{\lambda, a, b, aa, ab, bb, aaa, \cdots\}$$

以及

$$V^+ = \{a, b, aa, ab, bb, \cdots\}$$

在结构文法中，用希腊字母$\alpha, \beta, \gamma, \delta$……表示由非终止符和终止符组成的链，$\alpha, \beta, \gamma, \delta$……都属于 V_T 和 V_N 的闭包 $(V_T \cup V_N)^*$

$$\alpha, \beta, \gamma, \delta, \cdots \in (V_T \cup V_N)^* \tag{6-30}$$

当由一个链变到另一个链时，就是“产生式”P：

$$\alpha \to \beta \tag{6-31}$$

更广泛地讲，P 是产生式地有限集合。S 则表示起始符，它也属于 V_N。

由文法 G 所产生的语言 L 是由句子所组成，句子用小写英文字母 w，x，y，z 表示。整个语言用 $L(G)$表示，可写成

$$L(G) = \{x \mid x \in V_T^* 及 S \Rightarrow x\} \tag{6-32}$$

式中，x 表示由字母表组成的句子集合，属于V_T^*，而且 x 是由起始符 S 所生成的。

一般根据加于产生式的约束形式（根据产生式的不同形式），对文法进行分类。它有下列 4 种类型.

（1）上下文敏感文法

其产生式的一般具有下列形式：

$$\alpha_1 A \alpha_2 \to \alpha_1 B \alpha_2 \tag{6-33}$$

其中 $A \in V_N$，α_1，α_2，$B \in (V_N \cup V_T)^*$，$B \neq \lambda$。

（2）上下无关文法

其产生式的形式如下：

$$A \to B \tag{6-34}$$

其中 $A \in V_N$，$B \neq \lambda$，$B \neq A$。之所以称为上下文无关文法，是因为不考虑出现 A 的上下文，就可用链 B 代替变量 A。

（3）有限状态文法

其产生形式为：

$$A \to aB(或A \to Ba) \tag{6-35}$$

$$A \to a \tag{6-36}$$

其中 A，$B \in V_N$，$a \in V_T$。

（4）无约束文法

对产生不作限值的文法称为无约束文法。

对于一个句子 $x \in V_T^*$，一个由无约束文法产生的语言，不能对它作出决策。也就是说，

在句法结构的图像识别中，只有上下文有关文法、上下文无关文法及有限状态文法才是可利用的，因为它们在一个有穷的运算步骤内，可以判定一个模式是否属于一个确定的类别。而且，在图像识别中，经常采用的是有限状态文法和上下文无关文法，因为在这两个文法中，判定一个句子是否可以由相应的文法产生，是特别简单的。

【例 6-3】上下文无关文法 $G=(V_N,V_T,P,S)$用来描述亚中央染色体和远端染色体进行描述。

图 6-13 中所示的染色体图像基元是 G 的终结符，注意圆弧结构基元顺时针和逆时针的方向性。这里把染色体的轮廓沿顺时针方向编码成基元的连续连接，这样就把染色体表示成与基元的连接相对应的，在$\{a,b,c,d,e\}^+$之中的串。为了对染色体进行分类，把给定的串和 $L(G)$中的句子相匹配。如果匹配，则染色体就可以被识别为亚中央染色体或远端染色体，如果不匹配，则染色体就不能被这个文法所识别。

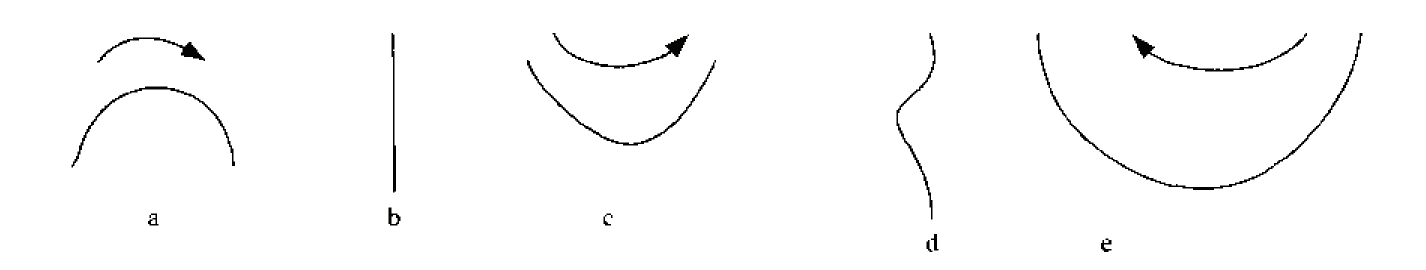


图 6-13 染色体文法基元

非终结符$\{S,S_1,S_2,A,B,C,D,E,F\}$，其中 S 是起始符。产生式集 P 由以下重写规则组成：

$S \rightarrow S_1$	$B \rightarrow e$	$S \rightarrow S_2$	$C \rightarrow bC$
$S \rightarrow AA$	$C \rightarrow Cb$	$S_2 \rightarrow BA$	$C \rightarrow b$
$A \rightarrow CA$	$C \rightarrow d$	$A \rightarrow AC$	$D \rightarrow bD$
$A \rightarrow DE$	$D \rightarrow Db$	$A \rightarrow FD$	$D \rightarrow a$
$B \rightarrow bB$	$E \rightarrow cD$	$B \rightarrow Bb$	$F \rightarrow Dc$

如果在 $L(G)$的句子导出中，第一个使用的产生式是$S \rightarrow S_1$，则这个句子代表亚中央染色体；如果使用的第一个产生式是$S \rightarrow S_2$，则这个句子代表远端染色体。因此，S_1代表<亚中央>，S_2代表<远端>。其余 6 个非终结符包含着染色体外形的物理特征，定义如下：A 表示<臂对>；B 表示<底>；C 表示<边>；D 表示<臂>；E 表示<右部>；F 表示<左部>。下面给出一个描述亚中央染色体的例子，其导出式是最左导出式：

$$S \Rightarrow S_1 \Rightarrow AA \Rightarrow ACA \Rightarrow FDCA \Rightarrow DcDCA \Rightarrow bDcDCA \Rightarrow bDbcDCA$$
$$\Rightarrow babcDCA \Rightarrow babcbDCA \Rightarrow babcbDbCA \Rightarrow babcbabCA$$
$$babcbabdA \Rightarrow babcbabdAC \Rightarrow babcbabdDEC \Rightarrow babcbabdaEC$$
$$\Rightarrow babcbabdacDC \Rightarrow babcbabdacaC \Rightarrow babcbabdacad$$

句子包含 12 个基元，图 6-14 是它所表示的染色体。

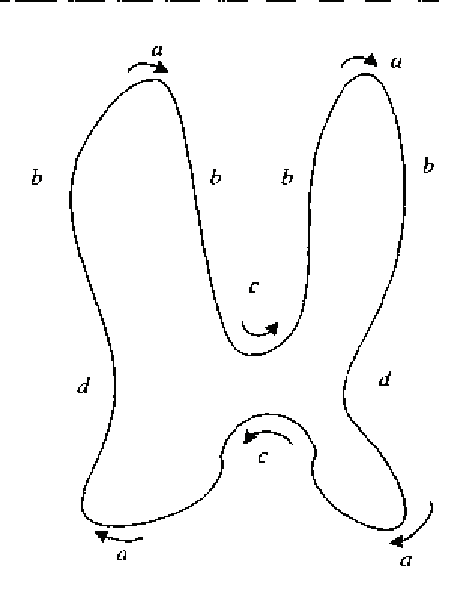


图 6-14　用句子 *babcbabdacad* 定义的亚中央染色体

2. 树文法

树状结构遵循树文法：

$$G_t=(V,r,P,S) \tag{6-37}$$

式中 $V=V_N\cup V_T$：V_T，r 是有限阶字母表，V_N 非终止符的有限集，，V_T 终止符的有限集，S 是起始符集合，P 是产生式规则集合：

$$\Phi\rightarrow\Psi \tag{6-38}$$

式中左端 Φ 是非终止符，右端 Ψ 是树。

由文法 G_t 产生的树状语言为：

$$L(G_t)=\{t\mid t\in T_{V_t},S\Rightarrow t\} \tag{6-39}$$

T_{V_T} 具有 V_T 中节点的树集。在 G_t 中，如果有产生式 $t_i\rightarrow t_j$，t_i 是 t_I 中在节点 a 处的子树，当能用 t_j 代替 t_I 产生出 t_II 时，记作：

$$t_\text{I}\overset{a}{\underset{G_t}{\Rightarrow}}t_\text{II} \tag{6-40}$$

【例 6-4】用树文法描述方块组成的图像，如图 6-15 所示。

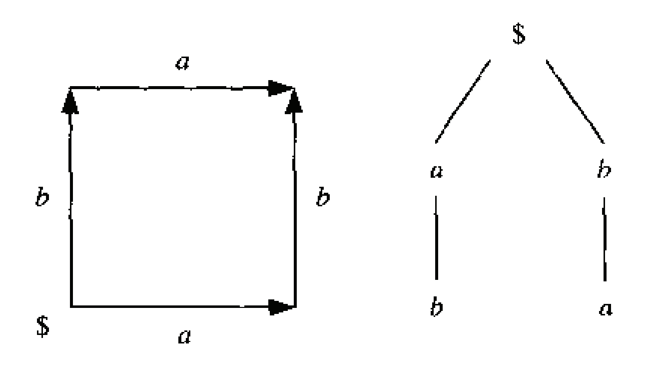


图 6-15　方块图像及其树状结构

$$G_t = (V, r, P, S)$$

其中$V = \{S, A, B, \$, a, b\}$，$V_T = \{a, b, \$\}$，$r(a)=\{2,1,0\}$，$r(b)=\{2,1,0\}$，$r(\$)=\{2\}$。产生式如图6-16所示，

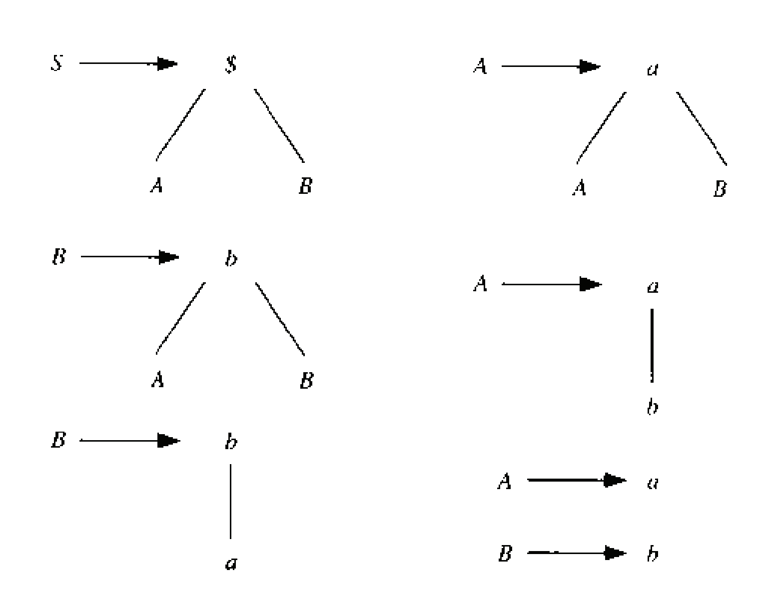


图6-16　树文法的产生式

在G中存在下面的导出：

$$S \underset{G_t}{\overset{\$}{\Rightarrow}} AB \underset{G_t}{\overset{a}{\Rightarrow}} bB \underset{G_t}{\overset{b}{\Rightarrow}} ba$$

得到与方块图像相对应的树状结构如图6-16所示

6.4　模糊集识别法

6.4.1　概述

"模糊"一词，译自英文"Fuzzy"，意为"模糊的"，"不分明的"。1965年美国控制论专家L.A.Zadeh首先将"Fuzzy"一词引入数学界，他在"Information and Control"杂志1965年第期中发表了"Fuzzy Sets"一文，标志着模糊数学的诞生。

人们在认识客观世界的过程中，常常遇到内涵和外延不确定的，模糊的概念。例如"班上个子高的同学"，"面积很大"……等都是具有模糊性的语言。而人类却又大都用这些模糊的描述来交流思想，互相通信，然后进行推理分析和综合评价，最后做出分类决策。模糊集理论就是研究如何利用模糊的信息对确定的事物进行定量分析。

精确性与模糊性的对立，是当今科学发展面临的一个十分突出的矛盾。各门学科迫切要求数学化、定量化，但科学的深入发展意味着研究对象的复杂化，而复杂的东西又往往难以精化。电子计算机的出现，在一定程度正在解决这个矛盾，要求高度的精确，但机器所执行日益繁杂的任务，往往无法实现高度的精确。例如，命令计算机从监视大厅的摄像镜头中找出长满大胡

子的高个子，如果程序在屏幕上提出问题：身长多少以上算大个子，或许你勉强可以回答，但若计算机又问：有多少根胡子以上算大胡子?你将会被这问题弄得啼笑皆非。

决不能将“模糊”两字看成消极的贬义词。过分的精确反而模糊，适当地模糊反而精确。人脑在计算速度、记忆能力等方面远逊于电脑。可电脑对事物的识别远不如人脑，其主要原因是电脑对模糊事物的识别和判决远不如人脑，上述的高个子大胡子，对于人脑来说是远不是什么困难的问题。这两个模糊特征是人所早已掌握好了的，只要把大厅中的人群按对此种特征的隶属程度作比较，即可迅速找到此人。

模糊数学诞生至今仅二十几年的发展历程，它在模式识别这一领域中的应用历史更为短暂，还远未成熟。本章中尽可能涉及模式识别的本质，并将重点放在隶属度函数的建立上。因为针对某一模式的识别，其难点也正在于此。尽管如此，对于某一特定的模式识别课题，仍没有似乎也不可能提供一种按部就班的通用解法，但更重要的是解决问题的思路，有了这一基础，就具备了解决具体问题的前提。

6.4.2　模糊集理论基础

1. 模糊集的概念

普通集合具有“非此即彼”的性质，要么属于某个集合，要么不属于某个集合。但是，如上所述的概念，“大胡子”，“高个子”等边界都不明确，也就无法用普通集合描述了。对此，1965 年 Zadech 提出了如下模糊集的定义。

【定义 6.1】设在论域 U 上的一个模糊集 $\underset{\sim}{A}$，是指：对于任意 $u \in U$，都确定了一个数 $\mu_{\underset{\sim}{A}}(u)$，称 $\mu_{\underset{\sim}{A}}(u)$ 为 u 对模糊集 $\underset{\sim}{A}$ 的隶属度，且 $\mu_{\underset{\sim}{A}}(u) \in [0,1]$。

映射：$\mu_{\underset{\sim}{A}}$：$U \to [0,1]$ 称为 $\underset{\sim}{A}$ 的隶属函数。为了方便起见，把 $\mu_{\underset{\sim}{A}}(u)$ 简记为 $\underset{\sim}{A}(u)$。

模糊集完全由其隶属函数所刻画。

当 $\mu_{\underset{\sim}{A}}$ 的值域＝{0,1}时，蜕化为一个普通子集的特征函数，$\underset{\sim}{A}$ 便蜕化成一个普通子集。普通子集是模糊集的特殊形态。

2. 模糊集合的表示

常用下面的方法来表示一个模糊集。

当 $U=\{u_1,u_2,\cdots,u_n\}$ 是有限集时，用下列方法表示 U 上的模糊集。

① Zadech 表示法

$$\underset{\sim}{A} = \frac{\underset{\sim}{A}(u_1)}{u_1} + \frac{\underset{\sim}{A}(u_2)}{u_2} + \cdots\cdots + \frac{\underset{\sim}{A}(u_n)}{u_n} \tag{6-41}$$

式中各项并不表示分数，而是元素 u_i 和它的隶属度函数 $\underset{\sim}{A}(u_i)$，“＋”号也不是求和，而是表示论域 U 的整体。论域中隶属度为 0 的项可以不写。

② 序偶表示法

$$\underset{\sim}{A}=\{(u_1,\underset{\sim}{A}(u_1)),(u_2,\underset{\sim}{A}(u_2))\cdots\cdots(u_n,\underset{\sim}{A}(u_n))\} \tag{6-42}$$

同样，论域中隶属度为 0 的项可以不写。

③ 向量表示法

排出 U 中各元素次序后，把各元素的隶属度按次序记为向量的形式，这时隶属度为 0 的项不能省略。

$$\underset{\sim}{A}=\{\underset{\sim}{A}(u_1),\underset{\sim}{A}(u_2)\cdots\cdots\underset{\sim}{A}(u_n)\} \tag{6-43}$$

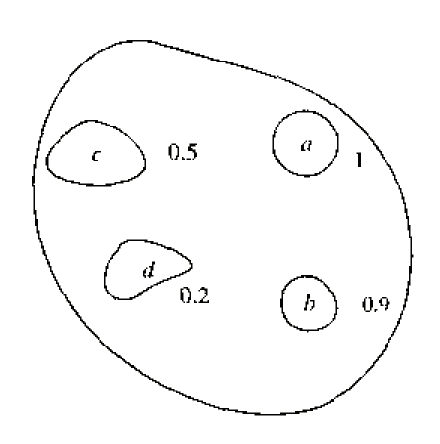


图 6-17　U 上的“圆形”模糊集

【例 6-5】如图 6-17 所示 $U=\{a,b,c,d\}$，对于 U 中的每一元素指定一个它对“圆形”的隶属度，设为 $\underset{\sim}{A}(a)=1$，$\underset{\sim}{A}(b)=0.9$，$\underset{\sim}{A}(c)=0.5$，$\underset{\sim}{A}(d)=0.2$，分别表征了它们对于“圆形”的隶属程度。

① Zadech 的表示法

$$\underset{\sim}{A}=\frac{1}{a}+\frac{0.9}{b}+\frac{0.5}{c}+\frac{0.2}{d}$$

② 序偶表示法

$$\underset{\sim}{A}=\{(a,1),(b,0.9),(c,0.5),(d,0.2)\}$$

③ 向量表示法

$$\underset{\sim}{A}=\{1,0.9,0.5,0.2\}$$

目前确定隶属函数的方法主要靠经验，从实践的效果不断反馈、调整自己的认识以达到预定的目标。但是，这种缺少理论化的判别原则，带有一定的盲目性。比较实用的确定隶属函数的方法有：①根据主观认识或经验，给出隶属度的具体数值；②根据问题的性质，选用某些典型函数作为隶属函数；③用统计调查结果得出的经验作为隶属函数。

【例 6-6】笔划类型的隶属函数

汉字中横、竖、撇和捺等线段的区分是根据它们与水平线的交角确定的。

设 A 为一线段，H，V，S 和 BS 分别为横、竖、撇和捺的 4 个模糊集合，则 A 对 4 个模糊

集的隶属度为：

$$\underset{\sim}{H}(A)=1-\min(\frac{|\theta|}{45},1) \qquad \underset{\sim}{V}(A)=1-\min(\frac{|90-\theta|}{45},1)$$

$$\underset{\sim}{S}(A)=1-\min(\frac{|45-\theta|}{45},1) \qquad \underset{\sim}{BS}(A)=1-\min(\frac{|135-\theta|}{45},1)$$

3. 模糊集的基本运算

【定义 6.2】设 $\underset{\sim}{A}$，$\underset{\sim}{B}$ 和 $\underset{\sim}{C}$ 是论域 U 上的 3 个模糊子集，若 $\forall u \in U$，有 $\underset{\sim}{C}(u)=\underset{\sim}{A}(u)\vee\underset{\sim}{B}(u)$，则称模糊集 $\underset{\sim}{C}$ 是 $\underset{\sim}{A}$ 与 $\underset{\sim}{B}$ 的并，记为 $\underset{\sim}{C}=\underset{\sim}{A}\bigcup\underset{\sim}{B}$。其中 $\vee$ 表示取大运算，即取两个隶属度中较大者作为运算结果。

若 $\forall u \in U$，有 $\underset{\sim}{C}(u)=\underset{\sim}{A}(u)\wedge\underset{\sim}{B}(u)$，则称模糊集 $\underset{\sim}{C}$ 是 $\underset{\sim}{A}$ 与 $\underset{\sim}{B}$ 的交，记为 $\underset{\sim}{C}=\underset{\sim}{A}\bigcap\underset{\sim}{B}$。其中 $\wedge$ 表示取小运算，即取两个隶属度中较小者作为运算结果。

若 $\forall u \in U$，有 $\underset{\sim}{B}(u)=1-\underset{\sim}{A}(u)$，则称模糊集 $\underset{\sim}{B}$ 是 $\underset{\sim}{A}$ 的补集，记为 $\underset{\sim}{B}=\overline{\underset{\sim}{A}}$。

【定义 6.3】设论域 U 上的模糊集 $\underset{\sim}{A}$，$0\leqslant\alpha\leqslant1$，称集合 $\underset{\sim}{A}_{\alpha}=\{u \mid \underset{\sim}{A}(u)\geqslant\alpha\}$ 为模糊集 $\underset{\sim}{A}$ 的 α—弱截集，简称 α—截集。α 称为置信水平，简称水平。把集合 $\underset{\sim}{A}_{\alpha}=\{u \mid \underset{\sim}{A}(u)>\alpha\}$ 称为模糊集 $\underset{\sim}{A}$ 的 α—强截集。

6.4.3　模糊关系

【定义 6.4】设 $U\times V$ 是集合 U 与 V 的直接积，称 $U\times V$ 的模糊子集 $\underset{\sim}{R}$ 为一个 U 到 V 的模糊二元关系，简称为模糊关系。隶属度 $\underset{\sim}{R}(u,v)$ 刻画了元素 u 与 v 关于 R 的相似程度。

设集合 $U=\{u_1,u_2,\cdots,u_m\}$，$V=\{v_1,v_2,\cdots,v_n\}$，$\underset{\sim}{R}$ 是 U 到 V 的一个模糊关系。矩阵 $\underset{\sim}{R}=(r_{ij})_{m\times n}$，使 $r_{ij}=\underset{\sim}{R}(u_i,v_j)$，称 $\underset{\sim}{R}$ 为模糊关系矩阵。简称模糊矩阵。

【例 6-7】用模糊关系表示苹果、乒乓球、书、足球、桃子、气球和四棱锥的相似关系，设用专家评分的办法给出它们相似程度如表 6-1。

表 6-1　　苹果、乒乓球等 7 种物品的相似程度

相似度	苹果	乒乓球	书	足球	桃子	气球	四棱锥
苹果	1	0.7	0	0.7	0.5	0.6	0
乒乓球	0.7	1	0	0.9	0.4	0.5	0
书	0	0	1	0	0	0	0.1
足球	0.7	0.9	0	1	0.4	0.5	0
桃子	0.5	0.4	0	0.4	1	0.4	0
气球	0.6	0.5	0	0.5	0.4	1	0
四棱锥	0	0	0.1	0	0	0	1

其相似程度的模糊关系矩阵为：

$$\underset{\sim}{R}=\begin{bmatrix}1 & 0.7 & 0 & 0.7 & 0.5 & 0.6 & 0\\ 0.7 & 1 & 0 & 0.9 & 0.4 & 0.5 & 0\\ 0 & 0 & 1 & 0 & 0 & 0 & 0.1\\ 0.7 & 0.9 & 0 & 1 & 0.4 & 0.5 & 0\\ 0.5 & 0.4 & 0 & 0.4 & 1 & 0.4 & 0\\ 0.6 & 0.5 & 0 & 0.5 & 0.4 & 1 & 0\\ 0 & 0 & 0.1 & 0 & 0 & 0 & 1\end{bmatrix}$$

【定义 6.5】设 $\underset{\sim}{R}$，$\underset{\sim}{S}$ 是直接积 $U\times V$ 上的模糊关系，对于任意$(u,v)\in U\times V$，模糊关系之间的运算定义如下。

（1）包含：$\underset{\sim}{R}\subseteq\underset{\sim}{S}$，即 $\underset{\sim}{R}(u,v)\leqslant\underset{\sim}{S}(u,v)$

（2）相等：$\underset{\sim}{R}=\underset{\sim}{S}$，即 $\underset{\sim}{R}\subseteq\underset{\sim}{S}$，且 $\underset{\sim}{S}\subseteq\underset{\sim}{R}$

（3）并：$\underset{\sim}{R}\bigcup\underset{\sim}{S}$，即 $(\underset{\sim}{R}\bigcup\underset{\sim}{S})(u,v)=\underset{\sim}{R}(u,v)\vee\underset{\sim}{S}(u,v)$

（4）交：$\underset{\sim}{R}\bigcap\underset{\sim}{S}$，即 $(\underset{\sim}{R}\bigcap\underset{\sim}{S})(u,v)=\underset{\sim}{R}(u,v)\wedge\underset{\sim}{S}(u,v)$

（5）补：$\overline{\underset{\sim}{R}}$，即 $\overline{\underset{\sim}{R}}(u,v)=1-\underset{\sim}{R}(u,v)$

【定义 6.6】设 $\underset{\sim}{R}$ 是 U 到 V 的模糊关系，$\underset{\sim}{S}$ 是 V 到 W 的模糊关系。则它们的合成 $\underset{\sim}{R}\circ\underset{\sim}{S}$ 是 U 到 W 的一个模糊关系，其隶属函数为：

$$(\underset{\sim}{R}\circ\underset{\sim}{S})(u,w)=\underset{v\in V}{\vee}(\underset{\sim}{R}(u,v)\wedge\underset{\sim}{S}(v,w)) \tag{6-44}$$

【定义 6.7】给定一个 $U \times U$ 的模糊关系 $\underset{\sim}{R} = (r_{ij})_{n\times n}$，$\underset{\sim}{R}$ 具有：

（1）自反性，若对于 $\forall u \in U$，$\underset{\sim}{R}(u,u) = 1$；

（2）对称性，若对于 $\forall u,v \in U$，$\underset{\sim}{R}(u,v) = \underset{\sim}{R}(v,u)$；

（3）传递性，即 $\underset{\sim}{R} \circ \underset{\sim}{R} \subseteq \underset{\sim}{R}$，

把具有自反性、对称性和传递性的模糊关系 $\underset{\sim}{R}$ 称为是 U 上的一个模糊等价关系。而把具有自反性和对称性的模糊关系 $\underset{\sim}{R}$ 称为是 U 上的一个模糊相似关系。

6.4.4 最大隶属原则识别方法

设 U 为被识别对象构成的论域，$\underset{\sim}{A}_1, \underset{\sim}{A}_2, \cdots, \underset{\sim}{A}_n$ 是 U 上的 n 模糊集。$u_0 \in U$ 是一个确定元素，若有：

$$\underset{\sim}{A}_i(u_0) = \max_{1 \le k \le n}\{\underset{\sim}{A}_k(u_0)\} \tag{6-45}$$

则判定元素 u_0 在 U 上应识别为 $\underset{\sim}{A}_i$ 类。这一原则称为最大隶属度原则。

当模式的隶属度确定之后，采用这种方法进行识别是比较简单的。关键是要准确地确定隶属度函数，如果隶属度确定不恰当，则识别的结果往往也不够理想。

最大隶属原则在图像识别中是很有用的，不过它只适于处理较为简单的问题。如果待识别模式并非某一个特定的元素，而是论域 U 中的一个模糊子集，用最大隶属原则就难以处理了，需要采用下述的方法。

6.4.5 择近原则识别方法

在实际应用中，有时待识别的对象并不是确定的单个元素，而是论域 U 上的模糊子集，并且已知的模式也是论域上的模糊子集。识别需要处理的是两个集合的问题。

【定义 6.8】设 $\underset{\sim}{A}$，$\underset{\sim}{B}$ 是论域 U 上的两个模糊集，若 $U=\{u_1,u_2,\cdots,u_n\}$ 是有限集合，定义：

$$M(\underset{\sim}{A},\underset{\sim}{B}) = \left[\frac{1}{n}\sum_{i=1}^{n} |\underset{\sim}{A}(u_i) - \underset{\sim}{B}(u_i)|^p\right]^{\frac{1}{p}} \tag{6-46}$$

p 是正实数。称 $M(\underset{\sim}{A},\underset{\sim}{B})$ 为模糊集 $\underset{\sim}{A}$，$\underset{\sim}{B}$ 之间的相对 Minkowski 距离，当 $p=1$ 时，称 $M(\underset{\sim}{A},\underset{\sim}{B})$ 为相对 Hamming 距离，记为 $M_H(\underset{\sim}{A},\underset{\sim}{B})$。

$$M_H(\underset{\sim}{A},\underset{\sim}{B})=\frac{1}{n}\sum_{i=1}^{n}|\underset{\sim}{A}(u_i)-\underset{\sim}{B}(u_i)| \tag{6-47}$$

当 p=2 时，称 $M(\underset{\sim}{A},\underset{\sim}{B})$ 为相对 Euclid 距离，记为 $M_E(\underset{\sim}{A},\underset{\sim}{B})$。

$$M_H(\underset{\sim}{A},\underset{\sim}{B})=\sqrt{\frac{1}{n}\sum_{i=1}^{n}|\underset{\sim}{A}(u_i)-\underset{\sim}{B}(u_i)|^2} \tag{6-48}$$

相应地，$N_H(\underset{\sim}{A},\underset{\sim}{B})=1-M_H(\underset{\sim}{A},\underset{\sim}{B})$ 为 $\underset{\sim}{A}$ 与 $\underset{\sim}{B}$ 模糊集 Hamming 贴近度。$N_E(\underset{\sim}{A},\underset{\sim}{B})=1-M_E(\underset{\sim}{A},\underset{\sim}{B})$ 为 $\underset{\sim}{A}$ 与 $\underset{\sim}{B}$ 模糊集 Euclid 贴近度。贴近度用来衡量两个模糊集的“相近程度”。

设论域 U 上的 n 模糊集，$\underset{\sim}{A}_1,\underset{\sim}{A}_2,\cdots,\underset{\sim}{A}_n$，设 $N(\underset{\sim}{B},\underset{\sim}{A}_i)$ 是模糊集 $\underset{\sim}{B}$ 与 $\underset{\sim}{A}_i$ 的某个贴近度，若存在 i_0，使得：

$$N(\underset{\sim}{B},\underset{\sim}{A}_{i_0})=\max_{1\leqslant k\leqslant n}\{N(\underset{\sim}{B},\underset{\sim}{A}_k)\} \tag{6-49}$$

则判定模糊集 $\underset{\sim}{B}$ 识别为 $\underset{\sim}{A}_{i_0}$ 类。这一原则称为择近原则。

6.4.6 模糊聚类识别方法

一般的模式识别问题是将一个未知模式分到已知类别的各种模式的一类中去。但是，在实际问题中有时要将一些事物分类，而分类前又不知道要分成几类，这种分类称为聚类分析。它将比较接近的事物按一定的量度归为一类。

1. 基于模糊等价关系的聚类

【定理 6.9】设模糊关系矩阵 $\underset{\sim}{R}$ 是模糊等价关系，则对于任意的 $\alpha\in[0,1]$ 所截得的截集 R_α 也是等价关系。

根据上述定理可知，若 $\underset{\sim}{R}$ 是模糊等价关系，则对于给定的 $\alpha\in[0,1]$ 便可得到相应的普通等价关系 R_α，这意味着得到了一个 α 水平的分类。

【定理 6.10】若 $0\leqslant\alpha_1\leqslant\alpha_2\leqslant1$，则 R_{α_2} 所分出的每一类必是 R_{α_1} 的某一子类，并称 R_{α_2} 分类法是 R_{α_1} 分类法的“加细”。

根据定理 6.10，当 α 由 1 逐渐下降到 0 时，分类由细变粗，逐渐归并形成一个动态聚类图。

【例 6-8】设论域 $U=\{x_1,x_2,x_3,x_4,x_5\}$ 表示 5 个人脸部集合，U 上的模糊关系 $\underset{\sim}{R}$ 表示它们彼

此之间的相象关系：

$$\underset{\sim}{R}=\begin{bmatrix}1 & 0.8 & 0.6 & 0.1 & 0.2\\ 0.8 & 1 & 0.8 & 0.2 & 0.85\\ 0.6 & 0.8 & 1 & 0 & 0.9\\ 0.1 & 0.2 & 0 & 1 & 0.1\\ 0.2 & 0.85 & 0.9 & 0.1 & 1\end{bmatrix}$$

其中，元素 u_{ij} 表示第 i 个人与第 j 个人的脸部相象程度。因为对自身当然完全相象，故 u_{ii}=1。从 $\underset{\sim}{R}$ 矩阵中可看出它满足自反性和对称性，因此是一个模糊相似关系。但是，它不满足传递性，所以不是模糊等价关系。

如果直接用 $\underset{\sim}{R}$ 进行分类，认为“相似程度”大于 0.8 的为一类，则 x_1，x_2 为一类，x_2，x_5 为一类，但 x_1，x_5 的“相似程度”仅为 0.2，故 x_1，x_5 不属于同一类，结论相互矛盾，说明模糊相似矩阵不能直接用来分类。

一般来说，对于一个具有相似关系但不具有等价关系的模糊矩阵 $\underset{\sim}{R}$，可以构造一个新的模糊关系矩阵使之具有模糊等价关系。用 $\underset{\sim}{R}$ 自乘得 $\underset{\sim}{R}^2$ ……直到 $\underset{\sim}{R}^{2k}$ 是一个模糊等价关系为止。

本例中，求出 $\underset{\sim}{R}^2$：

$$\underset{\sim}{R}^2=\begin{bmatrix}1 & 0.8 & 0.8 & 0.2 & 0.8\\ 0.8 & 1 & 0.85 & 0.2 & 0.85\\ 0.8 & 0.85 & 1 & 0.2 & 0.9\\ 0.2 & 0.2 & 0.2 & 1 & 0.2\\ 0.8 & 0.85 & 0.9 & 0.2 & 1\end{bmatrix}$$

上式为一个模糊等价关系。根据不同α水平进行分类。

（1）当$0.9<\alpha\leqslant 1$时

$$R_{\alpha}^2=\begin{bmatrix}1 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 1\end{bmatrix}$$

此时共分为 5 类：即每个元素各成一类。

（2）当$0.85<\alpha\leqslant 0.9$时

$$R_\alpha^2=\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&1\\0&0&0&1&0\\0&0&1&0&1\end{bmatrix}$$

此时共分为 4 类：即$\{x_3, x_5\}$为一类，其他每个元素各成一类。

（3）当$0.8<\alpha\leqslant0.85$时

$$R_\alpha^2=\begin{bmatrix}1&0&0&0&0\\0&1&1&0&1\\0&1&1&0&1\\0&0&0&1&0\\0&1&1&0&1\end{bmatrix}$$

此时共分为 3 类：即$\{x_2, x_3, x_5\}$为一类，其他每个元素各成一类。

（4）当$0.2<\alpha\leqslant0.8$时

$$R_\alpha^2=\begin{bmatrix}1&1&1&0&1\\1&1&1&0&1\\1&1&1&0&1\\0&0&0&1&0\\1&1&1&0&1\end{bmatrix}$$

此时共分为 2 类：即$\{x_1, x_2, x_3, x_5\}$为一类，$\{x_4\}$为一类。

（5）当$0<\alpha\leqslant0.2$时

$$R_\alpha^2=\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&0\\1&1&1&1&1\end{bmatrix}$$

此时仅有一类，是“最粗”的分类。

2. 基于模糊相似关系的聚类

上例已经说明对于模糊相似关系矩阵需要改造成模糊等价关系才能进行正确分类。但是，多次合成运算将消耗很多时间，特别是当样本数目较大时，问题将变得更加突出。人们希望寻找基于模糊相似矩阵直接分类的方法，如最大树法。

设$G=(V,\underset{\sim}{R})$为模糊图，V 是顶点集，共有 n 个元素，$\underset{\sim}{R}=(r_{ij})$是 V 上的模糊关系，且 G 是连通的。最大树法：先选出顶点集中的某一个顶点 V_i，然后将 r_{ij} 按从大到小的顺序依次连边，标上权重，并要求不产生回路，直到 n 个顶点都被连通为止，这样就得到一棵最大树。由于具体的连法不同，最大树不是惟一的。

得到最大树后，对于任意给定的α，只要将 $r_{ij}<\alpha$的边砍断就可得到一个不连通的图，它的各个连通的分支就是α水平上的类。这种分类法与最大树的选择无关，因此当最大树不惟一时，不影响其分类结果。

【例 6-9】设有 3 个家庭，每家有 4～7 人，选每个人一张相片，共 16 张混放在一起，对相片两两比较，按相貌的相似程度进行分类，希望能把 3 个家庭区分开。16 张相片的相似矩阵见表 6-2。

表 6-2　　16 张相片的相似矩阵

r_{ij}	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1															
2	0	1														
3	0	0	1													
4	0	0	0.4	1												
5	0	0.8	0	0	1											
6	0.5	0	0.2	0.2	0	1										
7	0	0.8	0	0	0.4	0	1									
8	0.4	0.2	0.2	0.5	0	0.8	0	1								
9	0	0.4	0	0.8	0.4	0.2	0.4	0	1							
10	0	0	0.2	0.2	0	0	0.2	0	0.2	1						
11	0	0.5	0.2	0.2	0	0	0.8	0	0.4	0.2	1					
12	0	0	0.2	0.8	0	0	0	0	0.4	0.8	0	1				
13	0.8	0	0.2	0.4	0	0.4	0	0.4	0	0	0	0	1			
14	0	0.8	0	0.2	0.4	0	0.8	0	0.2	0.2	0.6	0	0	1		
15	0	0	0.4	0.8	0	0.2	0	0	0.2	0	0	0.2	0.2	0	1	
16	0.6	0	0	0.2	0.2	0.8	0	0.4	0	0	0	0	0.4	0.2	0.4	1

构造最大树：选顶点 i=1，依次连“13”标 r_{ij}=0.8 于其侧边，再连“16”标 r_{ij}=0.6，由“16”再连接“6”，r_{ij}=0.8……，依次下去得到一棵连通 16 个顶点的最大树，如图 6-18 所示。然后对最大树取α截集，即去掉那些 $r_{ij}<\alpha$的边，这样就可以将它截成互不连通的几棵子树。

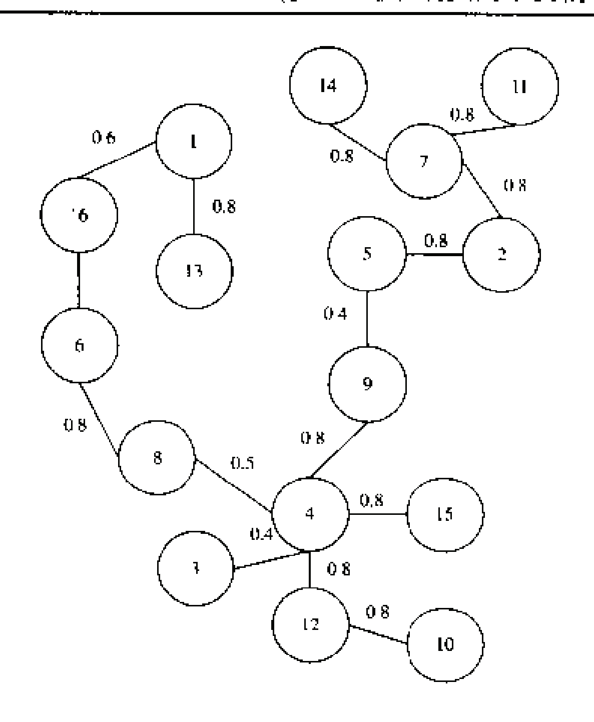


图 6-18　16 个顶点的最大树

现取α=0.6，可得到 4 颗子树，如图 6-19 所示，其顶点集为：

V_1={13，1，16，6，8}

V_2={9，4，15，12，10}

V_3={3}

V_4={11，7，14，2，5}

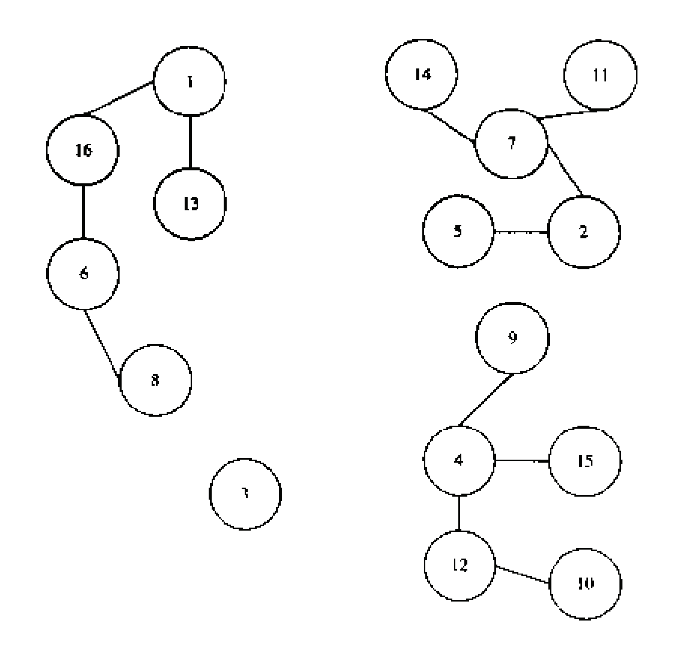


图 6-19　以α=0.6 切割图 6-18 后的结果

显然 V_1，V_2，V_4 符合每家 4～7 人的要求，而“3”不是这 3 个家庭的成员，实际上也是试验故意加进去的。应该指出，虽然最大树不惟一，但取了α截集，所得子树是相同的。

6.5　神经网络识别法

在模式识别的技术领域中，以特征提取为基础的方法遇到了极大的困难，如何表示和提取特征，需要多少特征，都存在很大的盲目性和低效率。识别过程必须经历从数据获取，特征提取到判决几个阶段，所需的运算使得系统难以满足实时性要求。为此，人们努力寻求新的理论和技术来解决这类问题，人工神经网络成为解决该类图像识别问题的有效工具。

6.5.1　人工神经网络概述

人工神经网络是指利用工程技术手段模拟人脑神经网络的结构和功能的一种技术，它的目的是使机器具有人脑那样的感知、学习和推理功能。人工神经网络之所以能取得巨大的发展，是因为它具有以下特点。

（1）神经网络具有分布式存储信息的特点。神经网络是用大量神经元之间的联结及对各联结权值的分布来表示特定的信息，因此即使当局部网络受损时，仍具有恢复原来信息的优点。

（2）神经网络对信息处理具有并行的特点。每个神经元都可以根据接收到的信息作独立的运算和处理，然后将结果传输出去，体现了一种并行处理的概念。

（3）神经网络对信息的处理具有自组织、自学习的特点。神经网络中各神经元之间的联结强度用权值大小来表示，这种权值可以事先给定，也可以适应周围环境而不断变化，这种过程称为神经元的学习过程。

神经网络用于图像识别具有以下优点：神经网络的信息分布式存储于连结权值系数中，使网络具有很高的容错性，而图像识别中往往存在噪声或输入图像的部分损失，因此神经网络可以较好地解决图像识别问题。另外，神经网络地自组织和自学习功能，大大放松了传统图像识别方法所需的约束条件，使其对图像识别问题显示出极大的优越性。

目前，神经网络用于图像识别的产品已经进入了商用阶段。比较成功的有 Nestor 和 NHC 公司的手写体识别，脑电图分析系统等。

6.5.2　与传统分类器的对比

现在介绍 6 种重要的用于模式分类的神经网络，并以此与传统模式分类器进行对比。与传统分类问题一样，现在假设它们均是用于将 n 维的样品分类归属 m 类模式中的某一类。

图 6-20(a)所示的传统分类器包含两级：第一级计算待识别样品（以下简称样品）对各类模式标本（以下简称样本）的匹配程度；第二级选出具有最大匹配度的类别。第一级的输入是用符号表示的 n 个输入元素的值，它们顺序地译码为有利于运算的内部形式。然后要设计出一种算法，以算出样品与每一类样本的匹配程度。显然每一样本应是该类模式的代表，而样品则往往是由样本以某种随机方式产生的。在这种情况下，总是假设样品的分布具有某种函数形式（例如正态分布函数等），因为这使匹配度的计算较简单。然后匹配度被顺序加载到分类器的第二级，并选出具有最大匹配度的类。

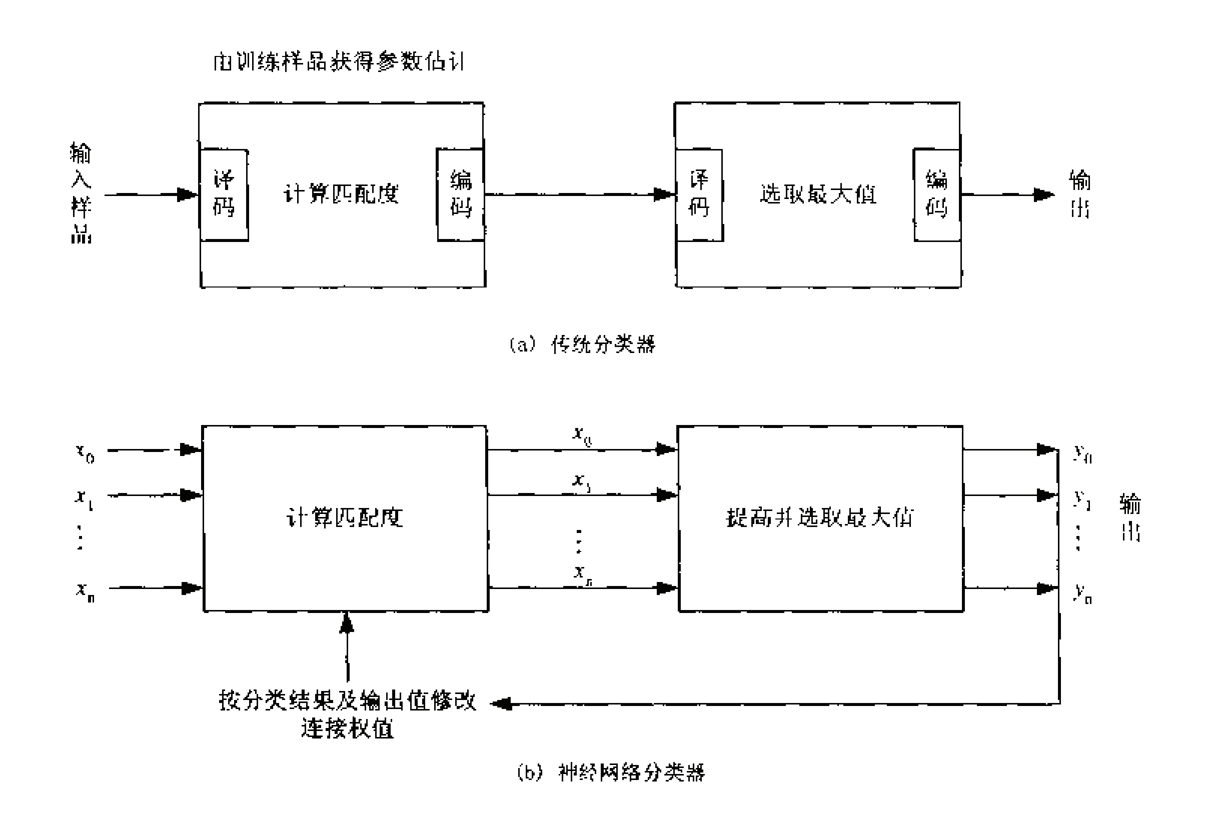


图 6-20 传统分类器与神经网络分类器的对比

图 6-20(b)是神经网络分类器。用 n 个分量表示的样品被送入神经网络，这些分量可用二值表示，或用连续值表示。神经网络的第一级实际上也是在计算匹配度，然后被平行地通过 m 条输出线送到第二级，在第二级中各类均有一个输出，并表现为仅有一个输出的强度为“高”，而其余的均为“低”。当得到正确的分类结果后，分类器的输出可反馈到分类器的第一级，并用一种学习算法修正权重。当后续的测试样品与曾学习过的样本十分相似时，分类器就会做出正确的响应。

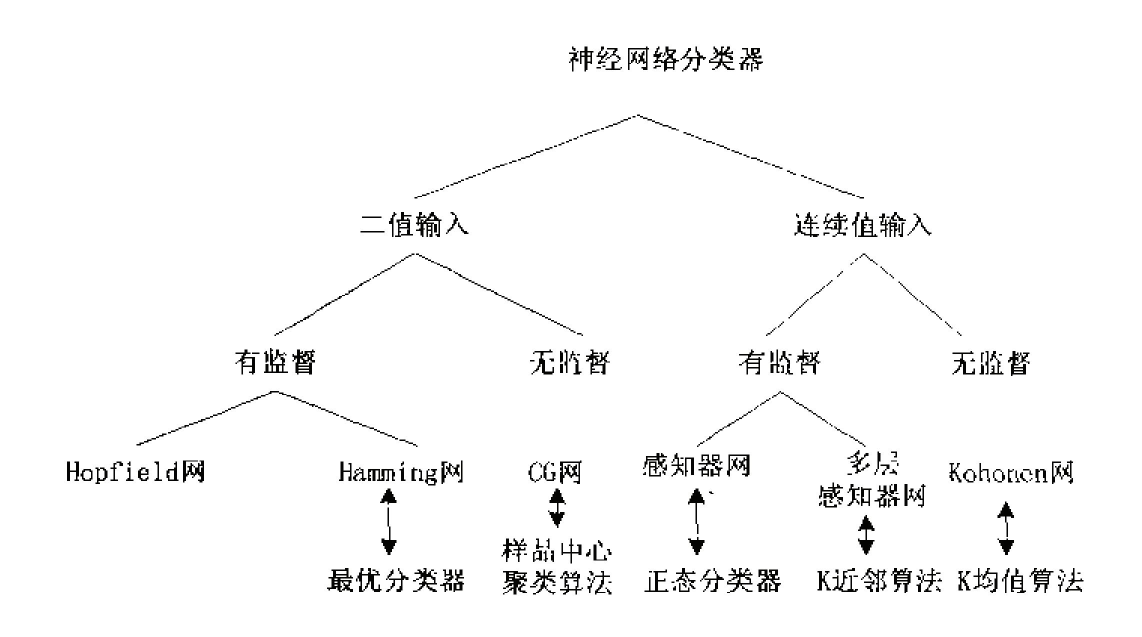


图 6-21 6 种神经网络分类器的分类树

图 6-21 所示给出这 6 种用于分类的神经网络的树状表示。第一层分枝是根据输入值为二

值变量还是连续变量；第二层分枝则是根据训练是否有监督。诸如 Hopfield 网，感知器网等是有监督训练的网，传统分类器（例如正态分类器等）都是有监督的，即所用的训练样本是已知类别的。而 Kohonen 网等则是无监督的。网络间的差异还在于是否支持自适应的训练，不过这一点在图 6-21 中未能体现出来。

图 6-21 下部还给出了与网络相对应的经典算法，这种对应关系有时是很好的，例如 Hamming 网确实与用于含有噪声的二值模式的最优分类器很相似。但有时却并非如此，例如感知器网与正态分类器的特性并不相同；Kohonen 网也并不执行迭代 K 均值训练算法，而是每一新模式加载后，权值就作相应修改。然而 Kohonen 网与 K 均值算法，在聚类数 K 的预先指定上则是一致的。

6.5.3 神经元模型

神经网络的处理单元称为神经元，也称为节点。图 6-22 所示为一个神经元模型的示意图。神经元的输入信号来自外部或其他神经元的输出。设输入信号分别为：$x_1,x_2,\cdots,x_n$，其中 n 为输入的数目；连接到神经元的权值相应为：$w_1,w_2,\cdots,w_n$。神经元兴奋的阈值为θ。则输入的加权和（也称激励电平）可以表示为：

$$u=\sum_{i=1}^{n}x_iw_i-\theta \tag{6-50}$$

激励电平通过激发函数 $f(\bullet)$ 的处理，得到神经元的输出为：

$$y=f(u)=f(\sum_{i=1}^{n}x_iw_i-\theta) \tag{6-51}$$

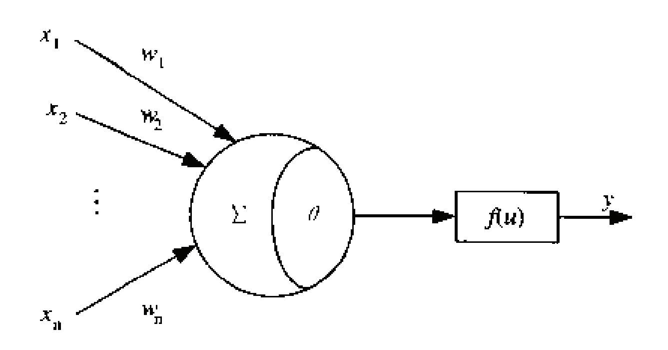


图 6-22 基本神经元模型

令 $x_0=-1$，$w_0=\theta$，（6-51）式可以表示为：

$$y=f(u)=f(\sum_{i=0}^{n}x_iw_i) \tag{6-52}$$

常用的激发函数如下。

（1）阈值型：输出为 0 或 1 两种状态。

（2）分段线性：$y=f(u)=\begin{cases}1 & u\geqslant r\\ \dfrac{u}{r} & |u|<r\\ 0 & u\leqslant -r\end{cases}$，有时也称为伪线性。

（3）S 型：$y=f(u)=\dfrac{1}{1+\exp(-u)}$。

上述介绍的人工神经元模型是生物神经元的一种近似，它在模拟生物神经网络时，已具备了生物神经元的某些特性，但也忽略了生物神经元的许多特性，如时间延迟等。

6.5.4 BP 神经网络分类器

神经网络的分类过程如图 6-23 所示。

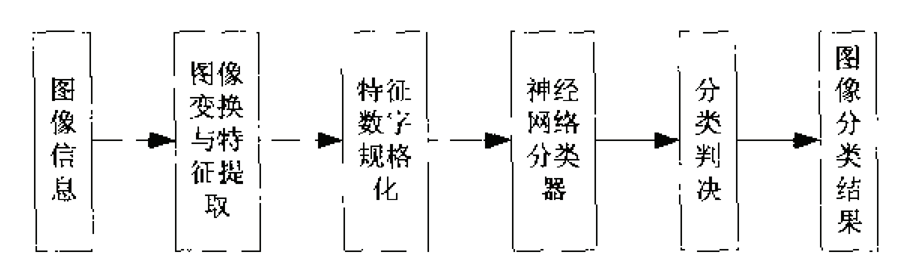


图 6-23 神经网络分类过程

1. BP 神经网络模型

通常所说的 BP 神经网络模型，即误差后向传播神经网络，是神经网络中使用最广泛的一类。如图 6-24 所示是 BP 神经网络的图像分类器：网络共分为 3 层：i 为输入层节点；j 为隐层节点；k 为输出层节点。层与层之间多采用全互连方式。同一层单元之间不存在相互连接。网络的每个输入节点表示图像特征向量的一个分量数据（灰度值），输出节点表示分类序号，分类判决可以采用输出最大值法。

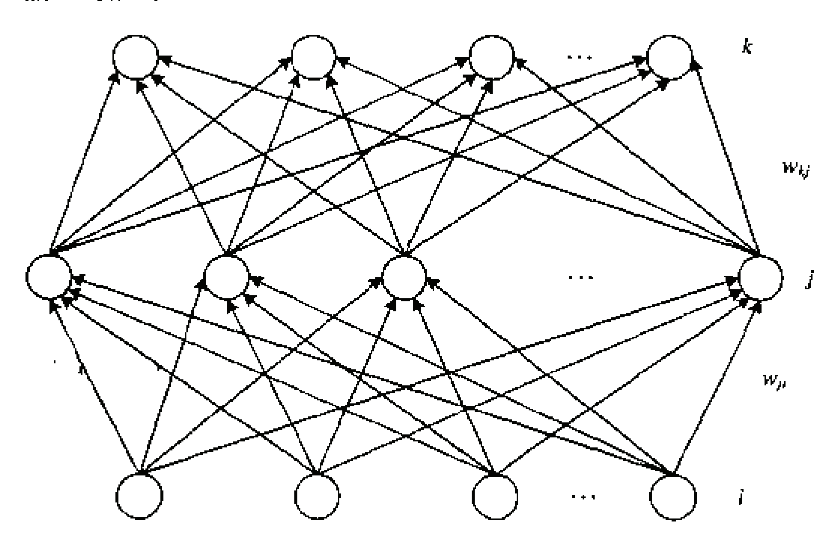


图 6-24 三层 BP 神经网络结构

BP 网络模型实现了多层网络学习的设想。当给定网络的一个输入模式时，它由输入层单元送到隐层单元，经隐层单元逐层处理后再送到输出层单元，由输出层单元处理后产生一个输出模式，故称为前向传播。如果输出响应与期望输出模式有误差，且不满足要求，那就转入误差后向传播，即将误差值沿连接通路逐层向后传送，并修正各层连接权值。

2. BP 网络学习算法

BP 网络学习是典型的有导师学习。训练集包含 M 个样本，对第 p 个训练样本（$p=1,2,\cdots,M$），单元 j 的实际输出为 O_{pj}，它的第 i 个输入（也即第 i 个神经元的输出）为 O_{pi}，则：

$$u_{pj}=\sum_{i=0}^{N}W_{ji}O_{pi} \tag{6-53}$$

BP 算法中大多选用 S 型函数作为输出函数，即：

$$O_{pj}=f(u_{pj})=\frac{1}{1+\exp(-u_{pj})} \tag{6-54}$$

定义网络误差函数为：

$$E=\sum_{p}E_{p} \tag{6-55}$$

$$E_{p}=\frac{1}{2}\sum_{j}(d_{pj}-O_{pj})^{2} \tag{6-56}$$

式中，d_{pj} 表示对 p 个训练样本，单元 j 的期望输出。训练网络的目的是找到一组权重，使误差函数极小化。

利用梯度最速下降法，使权值沿误差函数的负梯度方向改变。若权值的变化量记为 ΔW_{ij}，即：

$$\Delta W_{ij}\propto-\frac{\partial E_{p}}{\partial W_{ji}} \tag{6-57}$$

令 $-\dfrac{\partial E_{p}}{\partial u_{pj}}=\delta_{pj}$，则：

$$\frac{\partial E_{p}}{\partial W_{ji}}=\frac{\partial E_{p}}{\partial u_{pj}}\frac{\partial u_{pj}}{\partial W_{ji}}=\frac{\partial E_{p}}{\partial u_{pj}}O_{pi}=-\delta_{pj}O_{pj} \tag{6-58}$$

得到：

$$\Delta W_{ji}=\eta\delta_{pj}O_{pj},\quad \eta>0 \tag{6-59}$$

式中 η 为学习因子。

一般地，BP 网络学习算法步骤描述如下：

（1）初始化网络，将全部权值与节点阈值设置为一个较小地随机值；

（2）提供训练样本，训练网络，直到权值为各类样本均达到稳定。实际应用中，为了保证能正确识别各类对象，尽可能使准备的训练样本中包含了各类情况。

（3）前向传播过程：对给定训练模式输入，计算网络的输出模式，并与期望模式输出比较，若有误差，则执行（4），否则，返回（2）；

（4）后向传播，修正权值：

$$W_{ji}(t+1)=W_{ji}(t)+\eta\delta_{pj}O_{pj} \tag{6-60}$$

误差项δ_{pj}有两种情况：

$$\delta_{pj}=\begin{cases} f'(u_{pj})(d_{pj}-O_{pj}) & \text{输出单元} \\ f'(u_{pj})\sum\limits_{k}\delta_{pk}W_{jk} & \text{隐单元} \end{cases} \tag{6-61}$$

通常为了使学习因子η取值足够大，又不至于产生振荡，在权值修正公式（6-60）中再加一个势态项，即：

$$W_{ji}(t+1)=W_{ji}(t)+\eta\delta_{pj}O_{pj}+\alpha[W_{ji}(t)-W_{ji}(t-1)] \tag{6-62}$$

其中$0<\alpha<1$。

神经网络技术在文字识别、人脸识别等图像识别领域已有大量成功的应用。由于神经网络的学习需要大量样本，因此，在图像识别的学习过程中，需要大量的时间消耗。这里仅给出一个简单的神经网络应用实例，通过这个例子读者能够初步领会到神经网络的技术概念。

【例 6-10】利用神经网络技术通过对信号进行学习，从而预报下一时刻的信号。如图 6-25 所示为神经网络学习过程的曲线表示图形。

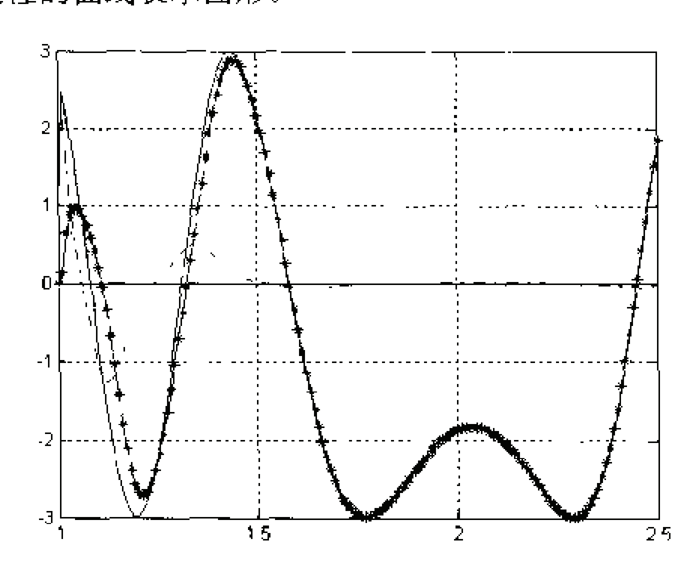


图 6-25　神经网络自适应学习过程及结果

```
time=1:0.01:2.5;
X=sin(sin(time).*time*10);
P=con2seq(X);
T=con2seq(2*[0 X(1:(end-1))]+X);  %创建一个信号 T
plot(time, cat(2,T{:}),'-');
Net=newlin([-3 3],1,[0 1],0.1); %生成一个神经网络；[-3 3]是预期的输入范围；第二个
                              %参数是每一层的神经元数目，[0 1]指定一个无延迟输入
```

```
                                        %和一个有延迟输入。最后一个参数表示学习速率。
[Net,Y,E,Pf]=adapt(Net,P,T);  %调用 adapt 函数来学习这个信号
figure,plot(time,cat(2,Y{:}),time,cat(2,T{:}),'-*',time,cat(2,E{:}),'-',[1
2.5], [0 0], 'k'); grid % 实线表示原始信号 T，*表示学习过程 Y，点划线表示学习误差，
                          % 如图 6-25 所示
```

相对于其他方法而言，利用神经网络来解决图像识别问题有 3 点优势：①神经网络对问题的先验知识要求较少；②可以实现对特征空间较为复杂的划分；③适合用高速并行处理系统来实现。

值得指出的是：对于大多数图像识别问题，由于同一类模式的图像样本往往存在着很大的差异，再考虑到图像样本的维数很高，所以虽然用神经网络进行直接识别在理论上是可行的，但是其时间的消耗是无法接受的，故通常先进行预处理完成特征提取，再由神经网络进行识别。

第7章 医学图像处理

医学图像处理就是将图像处理技术用于医学分析，如细胞的形态学分析、细胞的识别等，都是图像处理在医学上的重要应用领域。医学图像处理技术，有利于提高医学诊断的准确性、实效性。特别是模式识别技术与图像处理技术的结合，更增强了对细胞形态、病变等的自动分析能力，有助于制定客观、有效的治疗方案。

本章将介绍细胞边缘的精确检测以及癌细胞识别的图像处理。

7.1 细胞边缘的精确检测

7.1.1 概述

1. 问题的提出

比较同一种细胞在不同生理、病理或实验条件下形态的变化，可以为病理分析和病情诊断提供新的科学依据。目前的医学涂片判读过程基本上还是由人工完成的，由于细胞形态多样，变化微妙，数量又多，且交错分布，判读结果的准确性和效率受到很大的局限和影响。随着数学、计算机技术及医学等有关学科飞速发展，为细胞形态进行定量分析奠定了基础。利用计算机图像处理技术，研制高效、正确的医学涂片自动判读系统，已成为众多学者研究的热点。

细胞边缘的检测是进行细胞面积、圆度和个数等形态的定量计算和分析的基础。其检测结果直接影响病情的分析和诊断的结果，如果边缘检测不理想，根本就无法进行细胞的形态分析。经典的边缘提取方法是通过对图像的每个像素邻域内灰度的变化，即利用边缘邻近区域的一阶或二阶方向导数变化规律来检测边缘。Sobel 算子即是基于一阶方向导数在边缘处取最大值的变化规律来提取边缘的，而 Laplacian 算子则是基于二阶导数在边缘处过零的特点来提取边缘的。虽然这些算子计算简单、速度较快，但都存在如下缺陷：对噪声的干扰都很敏感，导致检测结果不稳定；得到的边缘像素是孤立的或分小段连续的，使得细胞面积和圆度的定量计算无法进行；它们检测所得到的边缘宽度比实际的大，引起相邻细胞边缘的重叠；在噪声较大时，这些算子无法检测细胞可靠的边缘位置，因而宜应用于细胞形态分析。针对上述问题，本文提出了改变传统边缘检测算法的处理顺序，利用最佳阈值分割和轮廓提取相结合的方法实现细胞真实边缘的检测，有效抑制噪声干扰的影响，保证了细胞边缘图像的连续性、完整性和精确定位。

2. 传统边缘检测算法存在的问题

设一幅细胞原始的数字图像用矩阵$[N,M]$表示，假设$f(i,j)$点受到噪声污染后变为$z(i,j)$，为便于分析，设$1\leqslant i\leqslant N-2$，$1\leqslant j\leqslant M-2$，则$f(i,j)$的3×3邻域为：

$$\begin{bmatrix} f(i-1,j-1) & f(i-1,j) & f(i-1,j+1) \\ f(i,j-1) & z(i,j) & f(i,j+1) \\ f(i+1,j-1) & f(i+1,j) & f(i+1,j+1) \end{bmatrix}$$

传统的边缘检测算子都采用梯度思想，即边缘点的求取是通过相邻几点的灰度运算得到的。以 Roberts 梯度算子为例，像素点（m,n）的梯度计算公式为：

$$\Delta_x f = f(m,n) - f(m-1,n-1) \tag{7-1}$$

$$\Delta_y f = f(m-1,n) - f(m,n-1) \tag{7-2}$$

$$\Delta f = \sqrt{(\Delta_x f)^2 + (\Delta_y f)^2} \tag{7-3}$$

即像素点（m,n）的梯度值与 $f(m,n)$，$f(m-1,n-1)$，$f(m-1,n)$，$f(m,n-1)$ 4 点的灰度值有关。这就意味着如果有一个点 $f(i,j)$ 被噪声干扰了，那么通过梯度运算检测边界时，与其相邻的 8 个点的梯度运算结果均受噪声的干扰，即边缘检测时噪声的影响通过检测算子被扩散。

梯度计算完成后，将结果与选取的门限值作比较，并做出判断，如果大于门限值，则该点为边缘点，否则不是边缘点。由于梯度计算结果，受噪声干扰严重，因此用梯度计算来检测边缘的方法对噪声敏感，检测结果不理想。而且，门限值的选取有较大主观性，其值选取好坏直接影响检测结果。

7.1.2 细胞边缘的精确检测

1. 基本原理

在细胞边缘检测计算过程中，为了有效地抑制噪声的影响，同时能够客观地、正确地选取边缘检测的门限值，改变边缘检测的操作顺序。先通过迭代算法求得图像分割的最佳阈值，并将图像分为背景和目标两部分。通过阈值分割处理，既增强了图像的目标与背景的对比，增强了细胞边缘,又能准确提取细胞区域。然后再利用轮廓提取算法，挖去细胞内部像素点，最后剩余部分图像就是细胞的边缘，从而实现了细胞的边缘检测。这样通过对各像素点自身灰度值的分析和计算，判断该点是否为边缘点，避免了在边缘检测的数学计算过程中，使噪声干扰的影响进一步扩大，损坏细胞边缘图像。

2. 分割最佳阈值的迭代算法

阈值分割方法是把图像的灰度分成不同的等级，然后用设置灰度门限的方法确定欲分割物体的边界。当用阈值来分割目标与背景时，如果某一灰度值 g 是某图像的分割阈值，即小于 g 的灰度点将构成目标（不妨如此假设），而大于 g 的灰度点就构成背景。一般而言，如果灰度值 g 可以作为图像的一个阈值，那么它应该使按这个阈值划分目标和背景的错误分割的图像像素点数为最小。

如果前景物体的内部具有均匀一致的灰度值，并分布在另一个灰度值的均匀背景上，那么图像的灰度直方图应具有明显的双峰，可是在许多情况下，噪声的干扰使峰谷的位置难以判定或者结果不稳定。本文采用迭代算法，有效地消除或减少噪声对灰度门限值 g 的影响。

设有一幅混入噪声的图像 $g(x,y)$ 是由原始图像 $f(x,y)$ 和 $e(x,y)$ 叠加而成的，即：

$$g(x,y)=f(x,y)+e(x,y) \tag{7-4}$$

这里假设各点的噪声是互不相关的，且具有零均值，标准差为ε。通过阈值分割将图像分割为两部分，由于噪声是随机作用于图像的像素点上，则可以认为在分割出目标 g_1 和背景 g_2 图像上噪声干扰仍为 $e(x,y)$，即：

$$g_1(x,y)=f_1(x,y)+e(x,y) \tag{7-5}$$

$$g_2(x,y)=f_2(x,y)+e(x,y) \tag{7-6}$$

在迭代算法中，需要对分割出的图像分别求其灰度均值，则：

$$E\{g_1(x,y)\}=E\{f_1(x,y)+e(x,y)\}=E\{f_1(x,y)\} \tag{7-7}$$

$$E\{g_2(x,y)\}=E\{f_2(x,y)+e(x,y)\}=E\{f_2(x,y)\} \tag{7-8}$$

上式说明，随着迭代次数的增加，平均灰度值将趋向于真值。因此，用迭代算法求得的最佳阈值不受噪声干扰的影响。根据上述分析，迭代算法描述如下。

首先选择一个近似阈值作为估计值的初始值，然后进行分割，产生子图像，并根据子图像的特性来选取新的阈值，再用新的阈值分割图像，经过几次循环，使错误分割的图像像素点降到最少。这样做的效果好于用初始阈值直接分割图像的效果，阈值的改进策略是迭代算法的关键。算法的步骤如下所示。

（1）选择一个初始阈值的估算值 $T^0=\{T^k \mid k=0\}$，

$$T_0=\frac{Z_{\min}+Z_{\max}}{2} \tag{7-9}$$

式中，$Z_{\min}$，$Z_{\max}$ 分别表示图像中的最小和最大灰度值。

（2）利用阈值 T^k 把图像分割成两组，R_1 和 R_2，其中

$$R_1=\{f(x,y) \mid f(x,y)\geqslant T^k\} \tag{7-10}$$

$$R_2=\{f(x,y) \mid 0<f(x,y)<T^k\} \tag{7-11}$$

（3）计算区域 R_1 和 R_2 的灰度均值 Z_1 和 Z_2，其中

$$Z_1=\frac{\sum_{f(i,j)<T^k} f(i,j)\times N(i,j)}{\sum_{f(i,j)<T^k} N(i,j)} \tag{7-12}$$

$$Z_2=\frac{\sum_{f(i,j)>T^k} f(i,j)\times N(i,j)}{\sum_{f(i,j)>T^k} N(i,j)} \tag{7-13}$$

式中，$f(i,j)$是图像上(i,j)点的灰度值，$N(i,j)$是(i,j)点的权重系数，一般 $N(i,j)=1.0$。

（4）选择新的阈值 T^{k+1}。

$$T^{k+1} = \frac{Z_1 + Z_2}{2} \tag{7-14}$$

（5）如果 $T^k=T^{k+1}$，则结束，否则 $K=K+1$，转步骤（2）。

3. 二值图像的轮廓提取

经过图像分割后，原图变成二值图像，图像轮廓提取算法就变得非常简单。二值图像轮廓提取的算法就是掏空内部点，如果原图中有一点为黑，且它的 8 个相邻点都是黑色时，判定该点是图像的内部点，则将该点删除。最后，经过这样的算法处理后，一幅图像留下的点即是图像的轮廓，也就实现了图像的边缘检测。

在数学形态学运算中，腐蚀具有消除物体边界点的作用。结构元素取 3×3 的黑点块，腐蚀将使物体的边界沿周边减少一个像素。那么边缘检测实际上相当于用 3×3 块的 9 个点结构元素对原图进行腐蚀，再用原图像减去腐蚀的图像。

令 X 为图像，B 为结构元素，B_Z 表示结构元素 B 平移 Z 后的结果，B^S 代表结构元素 B 关于原点的对称集合。其数学表示如下：

$$B^S = \{-b; b \in B\} \tag{7-15}$$

则腐蚀的运算定义：

$$X\Theta B^S = \{Z; B_Z \subseteq X\} \tag{7-16}$$

数学形态学提取边界的算子如下：

$$ED(X) = X - (X\Theta E) \tag{7-17}$$

式中，E：3×3 的结构元素；$ED(X)$：代表图像 X 的边界。

该方法检测到的物体边缘宽度仅为一个像素，因此具有较高的定位精度。同时，二值化处理后的图像具有完整的轮廓，所以轮廓提取检测到的边缘具有连续性。

4. 轮廓提取受噪声影响分析

根据噪声随机分布的特征，物体边界领域受噪声干扰的点，只有少部分幅度接近阈值 T 的像素点被漏检或误检。因此，受噪声干扰的领域点总体上不会影响边界的检测。

经过最佳阈值 T 的阈值分割后，噪声对图像干扰的范围大大减少。设图像中感兴趣的目标的像素点灰度分布的均值和标准差为 μ_1 和 σ_1，背景点灰度的均值和标准差为 μ_2 和 σ_2。噪声 $e(x,y)$ 互不相关，且具有零均值，标准差为 ε。则噪声的幅度分布概率为：

$$P(x) = \frac{1}{\sqrt{2\pi}\varepsilon}\int_{-\infty}^{x} e^{-\frac{x^2}{2\varepsilon^2}} dx \tag{7-18}$$

$$P(-3\varepsilon \leqslant x \leqslant 3\varepsilon) = \frac{1}{\sqrt{2\pi}\varepsilon}\int_{-3\varepsilon}^{3\varepsilon} e^{-\frac{x^2}{2\varepsilon^2}} dx = 99.7\% \tag{7-19}$$

噪声幅度分布以 99.7%的高概率集中在[-3ε,$+3\varepsilon$]范围内。

噪声干扰而引起错误分割的图像分为两部分。

（1）目标错分为背景部分。原来属于目标内容的像素，灰度值为 $T-3\varepsilon \leqslant f(x,y) \leqslant T$，受噪声干扰后，像素灰度变为 $T \leqslant f(x,y)+e(x,y) \leqslant T+3\varepsilon$，此时 $e(x,y)$ 取 $[0^+,3\varepsilon]$。则目标范围内像素点被错分的概率为

$$P_1\{T-3\varepsilon \leq X \leq T\} = \frac{1}{\sqrt{2\pi}\sigma_1}\int_{T-3\varepsilon}^{T} e^{-\frac{(x-\mu_1)}{2\sigma_1^2}} dx \tag{7-20}$$

$$P_1 = \phi(\frac{T-\mu_1}{\sigma_1}) - \phi(\frac{T-3\varepsilon-\mu_1}{\sigma_1}) \tag{7-21}$$

由正态分布的概率曲线可知，固定范围长度为 3ε，则当 $T=\mu_1$ 时，式（7-21）的概率 P_1 取值最大。而迭代算法求得最佳分割阈值 $T=(\mu_1+\mu_2)/2$，则 $\mu_1<T<\mu_2$ 即：

$$P_{1\max} < \phi(0) - \phi(\frac{-3\varepsilon}{\sigma_1}) = \phi(\frac{3\varepsilon}{\sigma_1}) - 0.5 \tag{7-22}$$

（2）背景错分为目标部分。原来属于背景内容的像素灰度为，$T \leqslant f(x,y) \leqslant T+3\varepsilon$，受噪声干扰后 $T-3\varepsilon \leqslant f(x,y)+e(x,y) \leqslant T$，此时 $e(x,y)$ 取 $[-3\varepsilon,0]$。背景范围内的像素点被错分的概率：

$$P_{2\max} < \phi(\frac{3\varepsilon}{\sigma_2}) - \phi(0) = \phi(\frac{3\varepsilon}{\sigma_2}) - 0.5 \tag{7-23}$$

其他图像内容，虽然受噪声干扰，但经过阈值分割，都能正确地分割成目标和背景。因此，对噪声敏感的区域仅仅集中在 $T-3\varepsilon \leqslant f(x,y) \leqslant T+3\varepsilon$ 范围内。对于 $\varepsilon << \sigma$，则受噪声影响的像素点占整幅图像的概率非常小。当 $\varepsilon=0.01$，$\sigma_1=0.01\sigma_2$ 时，$P_{1\max}= P_{2\max}<1.2\%$。因此阈值分割大大减少了受噪声干扰的像素点数目。

边缘是指其周围像素灰度有阶跃变化的那些像素的集合，它存在于目标与背景之间。因此，边缘点是指它两边像素的灰度值有显著不同，即灰度值之差△较大。而噪声敏感的区域集中在 $T-3\varepsilon \leqslant f(x,y) \leqslant T+3\varepsilon$ 范围内，范围总长为 6ε。根据边缘点像素邻域灰度特性得：

$$6\varepsilon < \triangle \tag{7-24}$$

即图像的边缘点邻域远离噪声干扰的敏感区。

在边缘附近有极少部分的像素点（大约为 0.3%）受噪声干扰的幅度大于 $\pm 3\varepsilon$，噪声随机作用于像素点上，这些像素点的灰度值被大大增大或减小，引起错误分割，在目标内部出现孤立的假背景像素点，在背景内部出现孤立的假目标像素点。轮廓提取时，这些像素点会成为孤立的、不连续边界点提取出来。由于假边缘像素点数目少，而且孤立地、随机分布地存在，因此这将不会影响细胞真实边缘的检测。

综上所述，经过图像分割处理后，有效减少了噪声对图像的干扰范围，并使图像的边缘邻域像素点远离噪声干扰的敏感区，从而提高了图像边缘检测抗干扰能力。

【例 7-1】细胞边缘检测

原始图像如图 7-1 所示，Gauss-Laplace 边缘检测结果如图 7-2 所示，Sobel 边缘检测如图 7-3 所示，细胞边缘精确检测如图 7-4 所示。

MATLAB 程序代码:

```
blood = imread('blood1.BMP');
[x,y]=size(blood);                    % 求出图像大小
b=double(blood);
N =sqrt(100) * randn(x,y);            % 生成方差为 10 的白噪声
I=b+N;                                % 噪声干扰图像
for i=1:x                             % 实际图像的灰度为 0~255
   for j=1:y
      if (I(i,j)>255)
         I(i,j)=255;
      end
      if (I(i,j)<0)
         I(i,j)=0;
      end
   end
end
z0=max(max(I));                       % 求出图像中最大的灰度
z1=min(min(I));                       % 最小的灰度
T=(z0+z1)/2;
TT=0;
S0=0; n0=0;
S1=0; n1=0;
allow=0.5;                            % 新旧阈值的允许接近程度
d=abs(T-TT);
count=0;                              % 记录几次循环
while(d>=allow)                       % 迭代最佳阈值分割算法
   count=count+1;
   for i=1:x
      for j=1:y
         if (I(i,j)>=T)
            S0=S0+I(i,j);
            n0=n0+1;
         end
         if (I(i,j)<T)
            S1=S1+I(i,j);
            n1=n1+1;
         end
      end
   end
   T0=S0/n0;
   T1=S1/n1;
   TT=(T0-T1)/2;
   d=abs(T-TT);
   T=TT;
```

```
end
Seg=zeros(x,y);
for i=1:x
    for j=1:y
        if(I(i,j)>=T)
            Seg(i,j)=1;                % 阈值分割的图像
        end
    end
end
SI=1-Seg;              % 阈值分割后的图像求反，便于用腐蚀算法求边缘
se1=strel('square',3);              % 定义腐蚀算法的结构
SI1=imerode(SI,se1);               % 腐蚀算法
BW=SI-SI1;                         % 边缘检测
%======传统的边缘检测方法======%
I=uint8(I);
BW1=edge(I,'sobel', 0.11);
BW2=edge(I,'log', 0.015);
%===========图像显示==========%
figure(1);
imshow(I);title('Original')                          % 显示阈值分割的图像
figure(2);
imshow(BW2);title('Soble')                           % 显示新算法的边缘图像
figure(3)
imshow(BW1);title('Gauss-Laplace')
figure(4)
imshow(BW);title('New algorithm')
```

程序运行结果：

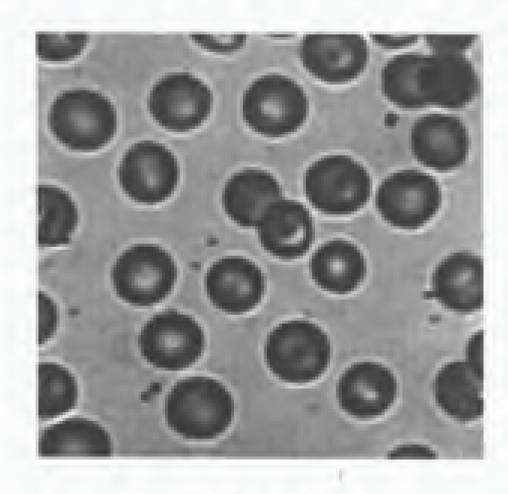


图 7-1　原始细胞图像



图 7-2　Gauss-Laplace 边缘检测

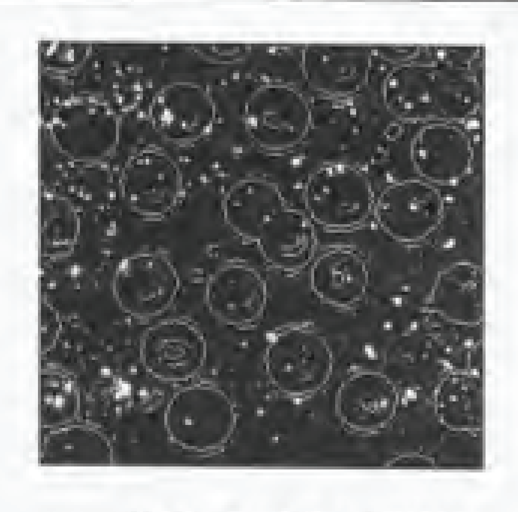


图 7-3　Sobel 算子边缘检测

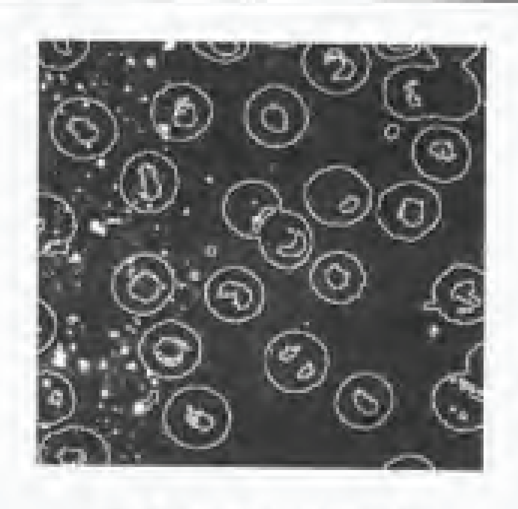


图 7-4　细胞边缘精确检测

7.1.3　算法总结

本节讲述了用迭代算法求图像分割最佳阈值和运用数学形态学的腐蚀算法实现轮廓提取相结合的细胞真实边缘精确检测算法，从理论上分析了该方法具有较强抗噪声能力，并用实验与传统的 Laplace 算子和 Sobel 算子等梯度边缘检测算子的检测结果进行了比较。证明该方法具有如下优点：

（1）细胞边缘检测取决于最佳分割阈值和像素本身的灰度值。迭代算法使最佳分割阈值不受噪声影响，因此边缘点完全由自身的灰度值确定，避免了噪声影响范围的扩大。

（2）细胞图像阈值分割能够提取完整的图像轮廓，使检测所得边缘具有连续性。

（3）数学形态学的腐蚀算法保证了检测得到的细胞边缘仅有一个像素的宽度，边缘定位精确。同时避免了边缘宽度增加而引起邻近细胞边缘的重叠。

（4）即使细胞的边缘比较模糊，也能通过阈值分割得到增强，运用腐蚀算法能可靠地提取边缘。

7.2　癌细胞识别系统

7.2.1　概述

长期以来，图像处理技术已经被广泛应用于各种医学应用领域中，其中许多是应用在微观医学与生物学中。医学涂片自动判读系统的研究，是国际上的难题和前沿课题。例如，国际上为了研究宫颈癌涂片自动判读系统，从 20 世纪 50 年代起，至少已投入了 4000 个人/年，其完成的效果为：可以去除样本中 50%的涂片，剩余 50%还需人工判读。因此研究这样的系统，有着重要的理论和实用价值。目前的图像诊断系统，大多数已使用了形态学、灰度特征和色度学，并结合专家系统，对癌细胞进行分析和诊断。

本文简单介绍了一个用于早期肺癌细胞普查的图像处理系统，该系统通过检查被检查者痰液涂片的彩色显微图像中是否存在癌细胞，以判断被测者是否患有肺癌。国内外的资料表明，

肺癌病例一经确诊 80%已属晚期，即失去手术治疗的机会。因此，及早地发现癌变，以达到准确的早日诊断和治疗已成为迫切需要解决的问题。目前，肺癌诊断的手段主要有：X 线胸片、CT、MRT、同位素、纤维支气管镜（BF）、经皮穿刺活检（PALB）、病理性诊断（如在痰中找脱落细胞）等，临床最可靠的还是病理性诊断，但病理性诊断的先进手段还相当缺乏。由于普查的工作量大，而人工判读受多种因素的制约，影响诊断的准确性与效率。因此，研制本系统的目的就是，利用计算机图像处理技术，减轻人的工作负担，提高诊断的准确性和效率。研制目标是在癌细胞识别率最高的前提下，假阳性率最小。由于所使用的涂片样本来自被检者的痰液，然后加以染色，和人体活检涂片相比，痰液中有较多的杂质和菌团，另外涂片染色可能不均匀，并且肺癌细胞种类不单一，主要有以下三种：磷癌细胞（SQ）、腺癌细胞（AD）和小细胞癌细胞（SM），因而，所处理的问题更加复杂，难度更大。针对这些情况，设计了一套切实可行的算法，在该系统中，通过把细胞核的形态学特征，以及色度学特征，同时用于癌细胞的识别，提高了系统的准确性，较好地完成了任务，性能指标优越。该系统的建立，不仅能使人从繁重的显微镜观察工作中解脱出来，提高诊断的准确性与效率，而且还能使该项肺癌早期诊断专家技术得以推广。本系统已接近专家的水平，它是彩色图像处理系统和专家系统的结合。

7.2.2 系统概况

系统硬件设置示意图如图 7-5 所示。在本系统中，痰液涂片的彩色细胞图像是通过连接于一个显微自动扫描平台的彩色摄像机实时获取的。该显微自动扫描平台由计算机自动控制，并可以在 x、y 和 z 3 个方向上自动定位和聚集。该系统所使用的痰液涂片是一种特殊的 PAT 痰细胞蓄积液溶解，并用 HE 染色法（Hematoxytin—Eosin 染色法）染色得到的。PAT 痰细胞蓄积液制作技术，是中国医科大学首创的一种技术，它为本系统的研究提供了良好的医学保证。为了提高系统对细胞分割与分类的准确性与效率，本系统采用了一种层次处理结构，系统处理示意图如图 7-6 所示。在该结构中，处理过程主要分为两个阶段：第一阶段是在一特定的彩色变换空间中利用自适应阈值方法的细胞核分割处理；第二阶段是分别利用细胞核的形态学特征及色度学特征对细胞核进行分类，以识别肺癌细胞的处理过程。

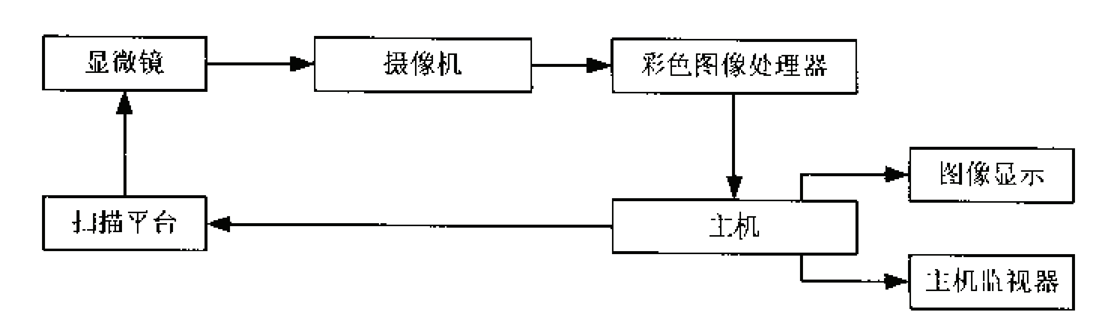


图 7-5 系统配置

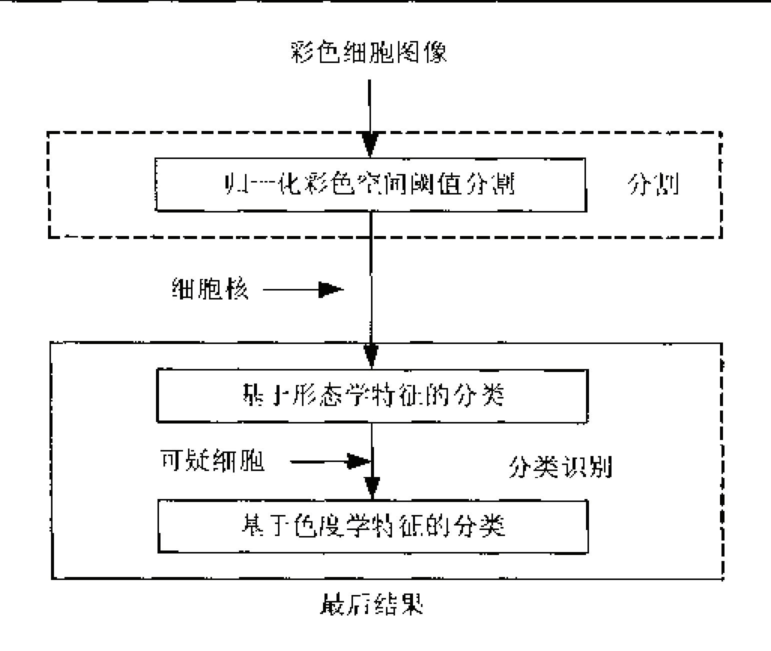


图 7-6 系统处理简要流程

7.2.3 阈值分割

1. 细胞核的分割

在本系统中，由于细胞核包含了识别肺癌细胞的主要特征，因此必须首先准确有效地将细胞核提取出来，以保证后继的识别工作能够正确地进行。但是，由于在痰片的制备过程中染色等条件的变化，以及在摄取彩色细胞图像时光照等条件的不同，所摄取的彩色图像中，细胞的彩色分量的分布将发生较大的变化。为此，必须选取某个特定的彩色空间，使得细胞核的特征在该空间中对染色及光照条件的变化不敏感，以保证细胞核的稳定提取。在本系统中，采用了归一化彩色空间技术和人工干预的半自动方法以进行细胞核的自动阈值分割。

2. 膨胀与腐蚀

在通过阈值分割获得的二值分割图像中，由于在核区内部可能存在小洞，在其周边存在凹坑和凸刺，使得其形态学特征的获取受到影响，因此，在本系统中，在提取细胞核形态特征之前，对其进行了适当的腐蚀与膨胀处理，以消除这些噪音的影响。这里采用了两次开方运算。实验表明，形态学开方运算提高了细胞核形态特征提取的准确性，从而提高了可疑癌细胞的识别准确率。

7.2.4 癌细胞识别

1. 形态学分类识别

一般地，在本系统中，肺癌细胞的细胞核比正常细胞的细胞核更大，在分割后的二值图像中，细胞核的边缘更加光滑、规则，形状更近似于圆或椭圆，因此在这里细胞核的形态学特征可以用来识别肺癌细胞。本系统采用了一种分层分类的结构，其中，形态学分类被用作分类处理的第一步：粗分类。因为经过粗分类，可以除去绝大多数正常细胞核，而只保留少量的可疑癌细胞作为后继分类与验证的对象，这样有效地提高了系统的识别效率。本方法中，使用细胞核主要的形态学特征包括细胞核的面积（A）、圆度（C）、矩形度（R）和伸长度（E）。在求取

以上 4 个形态特征时，本系统采用对二值分割图像进行边缘跟踪，求取其链码表示，从而获得其形态特征的方法。

（1）链码所围细胞区域的周长，即链的长度：

$$L = ne + no\sqrt{2} \tag{7-25}$$

其中，*ne* 表示链码中偶数码的数目；*no* 表示链码中奇数码的数目。

（2）细胞区域的圆度：

$$C = 4\pi A / L^2 \tag{7-26}$$

圆度用于描述细胞区域与圆形的偏离程度。在相同面积的条件下，细胞区域边界光滑且为圆形，则周长最短，其圆度为 $C = 1$。细胞区域的形状越偏离圆形，则 C 值越小。

（3）细胞区域的矩形度：

$$R = A / (W \times H) \tag{7-27}$$

其中，W 为宽度，H 为高度。矩形度用于描述细胞区域与矩形的偏离程度，当细胞区域为矩形时，R 取最大值 1。

（4）细胞区域的伸长度：

$$E = \min(W, H) / \max(W, H) \tag{7-28}$$

细胞区域越呈细长形，E 越小，当细胞区域为圆形时，$E = 1$。

当细胞核的形态学特征求得以后，利用下面所示的算法来选取可疑癌细胞：

```
For(每个细胞核)
  {
    if(Ath1<A<Ath2
    {
    if(C>Cth)并且(R<Rth) 并且(E>Eth)
    {该细胞为可疑癌细胞。}
    }
    else if(A>Ath2)
    {该细胞为可疑小细胞癌细胞。}
    else{该细胞为正常细胞。}
  }
```

式中，Ath1、Ath2、Cth、Rth、Eth 分别是 A、C、R 和 E 的阈值，它们是在训练阶段由实验获的。

以上算法的建立基于如下的事实：在本系统的条件下，肺癌细胞与正常细胞相比，其核区较大，且更近似于圆或椭圆。因此，当细胞核的面积大于某个下限值时（$A>\mathrm{Ath1}$），可以采用圆度、矩形度和伸长度约束（$C>\mathrm{Cth}$、$R<\mathrm{Rth}$ 和 $E>\mathrm{Eth}$）来选择可疑癌细胞。另外，由于小细胞癌具有聚堆的特点，使得在分割过程中很难将聚堆的小细胞癌的细胞分割为单个细胞，而只能将它们分割为一个“大细胞核”，并且其面积往往大于其他任何一种细胞核的面积，因此采用了另一个面积上限阈值（Ath2）来选择小细胞癌的细胞，即当某一细胞核的面积 A 大于阈值 Ath2 时，其被选作可疑小细胞癌细胞。

利用以上算法，进行了大量的形态学分类实验。从实验的结果可以看出，癌细胞的识别率

远高于非癌细胞的识别率，即绝大多数的癌细胞被正确地找出来了，而有许多的非癌细胞被错误地分类为癌细胞。实验表明，这主要取决于 Ath1、Ath2、Cth、Rth、Eth 的选择，而它们的选择决定于对癌细胞和非癌细胞识别率的折衷，即当非癌细胞的误识别率通过改变这些阈值而得到降低时，对癌细胞的识别率也会下降，反之亦然。因此，在本系统中，选择以上阈值的原则是，在保证癌细胞识别率最大的前提下，使非癌细胞误识别率最小，而在这一步的误分类被留给下一步的色度学分类去识别。

2. 色度学分类识别

在临床专家的指导下，对癌细胞核的色彩进行了分析研究。因为临床经验表明，和正常细胞相比，癌细胞的核颜色偏深，彩色分量有较大的差异，因而可以用细胞核的色彩特征进行分类识别。

由于色彩对光照是很敏感的，不同光照下色彩的变化是很大的，采集图像所用的彩色空间为 RGB 空间，而 R、G 和 B 这 3 个分量随光照的变化是不一致的，在实际使用过程中，很难要求系统每次的光照完全一致，因此很难找到一个理想的彩色聚类中心，为此采用了比较相对值的方法，这一方法是基于这样的事实，每一彩色分量的变化一般是一致的，因此计算 2×2 相对的比值更为合理，这样可以在很大程度上减小光照变化所带来的影响。通过对涂片分别在 RGB，LAB，HSI，YIQ 等彩色空间进行的大量实验，发现把 RGB，HSI 这两个彩色空间结合起来，识别效果较好，因此本系统采用了这两个彩色空间作为彩色识别空间。其中 H 表示色调，I 表示光照强度，S 表示饱和度。计算式分别为：

$$I=\frac{R+G+B}{3} \tag{7-29}$$

$$S=1-\frac{3}{R+G+B}[\min(R,G,B)] \tag{7-30}$$

$$H=\cos^{-1}\left[\frac{(R-G)+(R-B)}{2\sqrt{(R-G)^2+(R-B)(G-B)}}\right] \quad R\neq B \quad or \quad G\neq B \tag{7-31}$$

对于可疑癌细胞和正常细胞，分别计算了每个彩色分量的均值和方差，此外，还计算了 RGB 空间中 R 分量和 G 分量归一化后的差值 $DNBG$，因为通过实验发现，癌细胞的 $DNBG$ 具有较大的值，而正常细胞的该值较小，因而是一个很有用的特征参数，$DNBG$ 的计算公式如下：

$$2DNBG=(B-G)/I \tag{7-32}$$

$$I=0.3\times R+0.59\times G+0.11\times B \tag{7-33}$$

在此基础上，根据专家的经验，对这些统计量进行组合，归纳出 20 多条规则，并给每条规则赋以可信度，以此作为分类的依据。由于我们的系统是用于普查的，因此该系统必须满足以下两点要求：

（1）对癌细胞必须能识别出来，不能把癌细胞遗漏；

（2）不能把过多的非癌细胞误识别为癌细胞，即假阳性率太高也是不允许的。

为此，系统采用了序贯分析的方法，构造一个分类树，来满足上述要求，具体做法如下。

首先将由形态学分类出来的可疑癌细胞，根据一些显著的彩色特征，分为非癌细胞和仍然

可疑癌细胞两类，并确保分离出的非癌细胞中不包含癌细胞，然后对仍然可疑癌细胞进一步进行分类，当然可疑的癌细胞主要包括癌细胞，以及和癌细胞颜色相似的杂质和菌团等，根据色彩特征再一步地排除这些杂质和菌团，最终识别出癌细胞。在分类过程中，系统采用了专家系统中的似然推理技术，计算结论的可信度，根据可信度再得出最终的结论。

通过上述方法，可以比较满意地达到上述的两点要求，有利于提高系统的准确率和处理速度，也利于对系统规则进行修正。

【例 7-2】癌细胞形态学分析

癌细胞图像如图 7-7 所示，经人工辅助的二值化后的图像如图 7-8 所示，提取出的细胞周长如图 7-9 所示。

MATLAB 程序代码：

```
I=imread('cancer02.bmp');%注意必须保证二值图像中，细胞区域为白色区域或者像素点值为“1”
[x,y]=size(I);
BW = bwperim(I,8);                % 检测细胞的边缘跟踪，用于计算周长
%检测垂直方向连读的周长像素点%
P1=0;
P2=0;
Ny=0;                         % 记录垂直方向连续周长像素点的个数
for i=1:x
    for j=1:y
        if (BW(i,j)>0)
            P2=j;
            if ((P2-P1)==1)     % 判断是否为垂直方向连续的周长像素点
                Ny=Ny+1;
            end
            P1=P2;
        end
    end
end
%检测水平方向连读的周长像素点%
P1=0;
P2=0;
Nx=0;                          记录水平方向连续周长像素点的个数
for j=1:y
    for i=1:x
        if (BW(i,j)>0)
            P2=i;
            if ((P2-P1)==1)     % 判断是否为水平方向连续的周长像素点
                Nx=Nx+1;
            end
            P1-P2;
        end
    end
end
```

```
SN=sum(sum(BW));              % 计算周长像素点的总数
Nd=SN-Nx-Ny;                  % 计算奇数码的链码数目
H=max(sum(I));                % 计算细胞的高度
W=max(sum(I'));               % 图像 I 经矩阵转置后，计算宽度
L=sqrt(2)*Nd+Nx+Ny;           % 计算周长
%====4 个形态特征值计算===%
A=bwarea(I);                  % 计算细胞的面积
C=4*pi*A/(L*L);               % 计算圆度
R=A/(H*W);                    % 计算矩形度
E=min(H,W)/max(H,W);          % 计算伸长度
%==设定相关阈值，识别癌细胞==%
Ath1=10000; Ath2=50000;
Cth=0.5;     Rth=0.5;   Eth=0.8;
if ((A>=Ath1)&&(A<Ath2))
   if ((C>=Cth)&&(R<=Rth)&&(E>Eth))
      Cancer_Right=1                      % 结论为癌细胞
   end
else if (A>=Ath2)
      Cancer_Right=2                      % 结论为可疑小细胞癌细胞
else
   Cancer_Right=0                         % 结论为正常细胞
end
end
figure(1);
imshow(I);                    %如图 7-8 所示
figure(2);
imshow(BW);                   %如图 7-9 所示
```

程序运行结果：

```
Cancer_Right=1
```

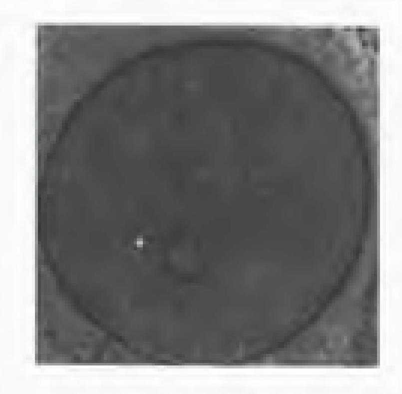


图 7-7　癌细胞涂片原始图像



图 7-8　二值化后的图像

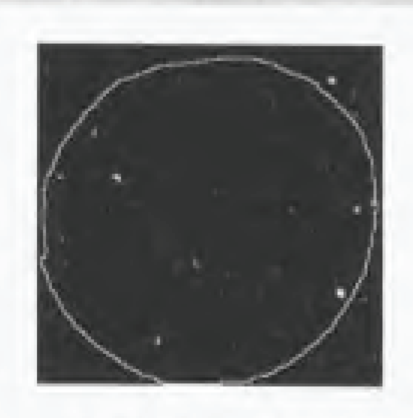


图 7-9　细胞核的周长线

【例 7-3】癌细胞颜色分析

原始图像 7-7 经颜色转换后的光分量图像如图 7-10 所示。

MATLAB 程序代码:

```
I=imread('cancer.bmp');
[y,x,z]=size(I);
myI=double(I);
% 图像的数据类型为无符号整型，必须转换为双精度实型才能计算
%====== RGB to HSI ======%
H=zeros(y,x);
S=H;
HS_I=H;
for i=1:x
    for j=1:y
        HS_I(j,i)=((myI(j,i,1)+myI(j,i,2)+myI(j,i,3))/3);
        S(j,i)=1-3*min(myI(j,i,:))/(myI(j,i,1)+myI(j,i,2)+myI(j,i,3));
 % 三者相等，H 计算式中分母为零，无法计算
        if ((myI(j,i,1)==myI(j,i,2))&(myI(j,i,2)==myI(j,i,3)))
            Hdegree=0;
        else
 Hdegree=acos(0.5*(2*myI(j,i,1)-myI(j,i,2)-myI(j,i,3))/((myI(j,i,1)-myI(j,i,
2))^2+(myI      (j,i,1)-myI(j,i,3))*(myI(j,i,2)-myI(j,i,3)))^0.5);
        end
        if (myI(j,i,2)>=myI(j,i,3))
            H(j,i)=Hdegree;                % G≥B, H 在[0,π]范围内
        else
            H(j,i)=(2*pi-Hdegree);          % G < B, H 在(π,2π)范围内
        end
    end
end
%=====色度识别====%
% 色度识别需要依靠专家知识，此处仅举一例说明
Hth1=0.9*2*pi; Hth2=0.1*2*pi;   % 检测红色，红色的色度值在 0 或 2π 附近
Nred=0;                          % 记录红色或接近红色的像素点个数
for i=1:x
    for j=1:y
        if ((H(j,i)>=Hth1)||(H(j,i)<=Hth2))
            Nred=Nred+1;
        end
    end
end
Ratio=Nred/(x*y);                % 红色像素点所占比例
if (Ratio>=0.6)                  % 判别是否为红色
    Red=1
else
```

```
    Red=0
end
HS_I=uint8(HS_I);                    % 实型转换为整型才能作为图像显示
figure(1);                          % 彩色原图如图 7-7 所示
imshow(I);
figure(2);
imshow(HS_I);                       % 光强分量的图像如图 7-10 所示
```

程序运行结果：

```
Red=1
```



图 7-10　光强分量图

第8章 文字图像识别

文字是人类相互交流信息的重要工具。与其他信息载体相比，它具有便于信息保存和传递的优点，使信息在时间和空间上得以迅速扩散。并且用简单的几个字符可以表达大量的信息，且能方便地被人们接受和理解。因此，文字的信息表达几乎渗透了人类活动的各个角落。社会发展进入信息时代，人们已不再停留在用自己的耳朵和眼睛去直接获得这些信息，再用手将信息记录在纸上，然后进行信息的分析处理。而是使用计算机代替人们简单、重复的劳动，将文字自动地输入计算机，用计算机对它们进行处理，随时以各种方式满足人们的不同需要。因此，研究如何用计算机自动识别文字图像，解决文字信息自动输入计算机，并进行高速加工处理的问题已引起大家的广泛关注。

在人们的日常生活中，在机关事务处理、工业以及商业交往中，需要识别文字的数量如同天文数字，但利用计算机识别的文字量只占需识别的文字量极少的百分比。最近几年，随着计算机技术、数学和图像技术的发展，文字识别的应用领域逐步扩大，目前较为活跃的应用包括数字图书馆中图书索书号的自动识别，公共交通管理中的汽车牌照自动识别以及旅游文化领域的商标自动翻译等。

8.1 文字图像识别简介

一般来说，文字图像的识别过程主要由以下4个部分组成：①正确地分割文字图像区域；②正确地分离单个文字；③正确识别单个文字；④正确地连接单个文字，即按照文字的排列顺序输出识别代码。其中①、④属于文字图像分析技术问题，③属于文字识别技术问题。关于②，由于仅从分割处理不能对其进行评价，通常采用文字识别的评价值来判断分离的正确性。单纯的文字识别是指经二值化处理后的单个文字识别。

8.1.1 文字识别系统的原理及组成

文字图像识别的原理框图如图8-1所示。图中光电变换检测部分的主要功能，是对纸面上的文字进行光电转换，然后经模数转换成具有一定灰度的数字信号，送往其后的各部分进行处理和识别。常用的检测设备是扫描仪，CCD 摄像头等。文字图像分割的目的就是根据文字图像的特征的实现文字图像区域的定位和分割，将真正的文字图形分割出来，以便后续进行识别。识别预处理部分的功能是将已分割出的文字图形信息加以区分，去除信号中的污点、空白等噪声，增强文字图像的信息。并根据一定的准则除掉一些非本质信号，对文字的大小、位置和笔划粗细等进行规范化，以便简化判断部分的复杂性。特征提取部分是从整形和规范化的信号中抽取反映字符本质的有用信息，供识别部分进行识别。作为特征提取的内容是比较多的，可以是几何特征，如文字线条的端点、折点和交点等。识别判断部分则是根据抽取的特征，运用

定的识别原理，对文字进行分类，确定其属性，达到识别的目的，实际上判断部分就是一个分类器。

识别系统学习部分的功能是生成计算机特征字典，学习根据已经准备好的多个字样，抽出代表该字的特征，进行修改，按照字典的规定位置存放该特征。学习分为两种：一种是在人的参与下进行，称为“有教师”学习；一种由计算机自动进行，称为“无教师学习”。

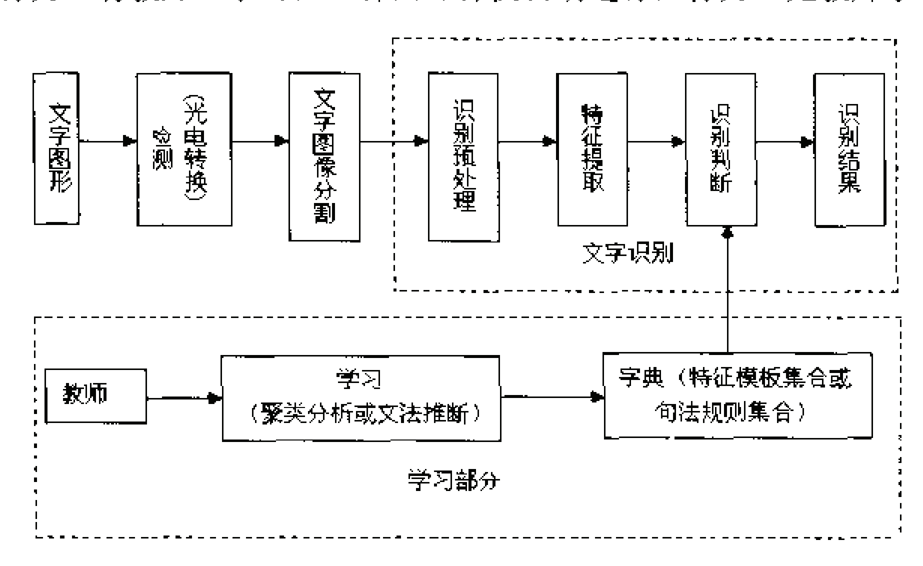


图 8-1　文字识别原理框图

8.1.2　文字识别的方法

本书所涉及的文字识别是指用计算机自动、高速地识别写在介质（如纸张等）上的数字、英文符号或汉字。文字识别实际上就是解决文字的分类问题，一般通过特征判别及特征匹配的方法来进行处理。

特征判别是通过文字类别（例如英文或汉字）的共同规则（如区域特征、四周边特征等）进行分类判别。它不需要利用各种文字的具体知识，根据特征抽取的程度（知识的使用程度）分阶段地使用结构分析的办法完成字符的识别。

匹配的方法则是根据各个文字的知识（称为字典）采取按形匹配的方法进行。按实现的技术途径不同又可分为两种：一种是直接利用输入的二维平面图像与字典中记忆的图像进行全域匹配；另一种是只抽出部分图像与字典进行匹配。然后根据各部分形状及其相对位置关系，与保存在字典中的知识进行对照，从而识别出每一个具体的文字。前一种匹配方法适合于数字、英文符号一类的小字符集；后一种匹配方法适用于汉字一类的大字符集。

匹配的方法一般适用于规范化的印刷文字，特别是同一字体的印刷文字。特征判别的结构分析法多用于手写文字的识别。一般来说，匹配方法的程序编制简单，字典占据空间大，识别速度高；结构分析方法程序复杂，能够处理手写体文字的变形问题，具有区分近似文字的优点，但将其用于初始分类则具有不稳定的缺点。所以，有时往往将两种方法结合起来使用。

8.2　图书馆中图书索书号的自动识别

索书号是图书馆赋予每一种馆藏图书的号码。这种号码具有一定结构并带有特定的意义。

在馆藏系统中，每种索书号是惟一的，可借以准确地确定馆藏图书在书架上的排列位置，是读者查找图书非常必要的代码信息。

信息社会的到来，使现代图书馆演变为重要的信息集散地。图书馆藏书根据图书索书号的正确摆放是保证读者能够及时查找图书的前提。但是，由于图书馆的藏书多，每天借阅图书的人流大，经常会发生图书摆放位置与索书号不一致的问题。目前主要依靠人工检查纠正，但是工作效率低，工作量大，耗费时间长，纠正不及时，造成图书资源的浪费。因此，研究图书索书号自动识别技术，对图书摆放位置是否正确进行自动判别具有重要意义。

8.2.1 索书号自动识别系统概述

系统的硬件由彩色 CCD 摄像头，8 位图像采集卡和计算机组成，拍摄的图像为 8 位 RGB 彩色数字图像。

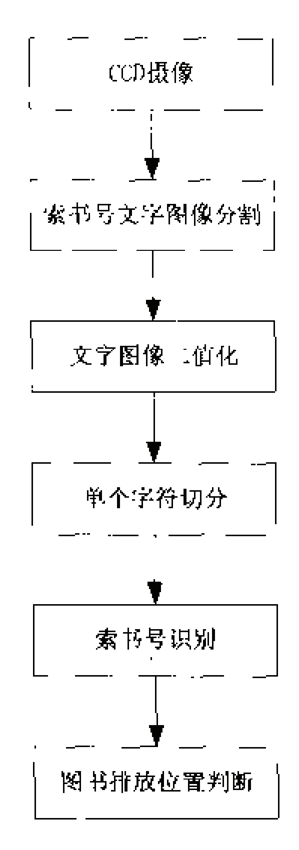


图 8-2　索书号识别流程

索书号自动识别流程框图如图 8-2 所示。CCD 摄像头拍摄排放在书架上的图书图像。图书的图像除所属号外还有其他内容，如出版社的标志，其他文字甚至大块的污斑等，必须从中分割出仅包含索书号的文字图像。灰度的文字图像难以直接用于识别，所以将灰度的文字图像转换为二值图像。由于噪声和污点的干扰，以及光照不均匀的影响，二值化后的文字会发生粘连，严重改变文字的字形。字符切分成单个字符后才进行文字识别。按照上述流程，具体介绍索书号识别的整个过程。

8.2.2 索书号文字图像分割

CCD 拍摄的图书索书号图像如图 8-3 所示。图像中除了索书号还包含有其他文字，红色

条形框以及装饰景物等，为了能够正确识别该书的索书号，首先必须从图像中正确分割出索书号图像。



图 8-3　图书索书号图像

然而，从图像尤其是场景图像中自动定位和分割文字是比较困难的，其主要原因有以下几个方面：①文字嵌入在图像中，并与其他图形共存，如边框，商标、装饰物以及污点等；②由于文字颜色退化，文字颜色不均匀，而且背景颜色有时与文字颜色差异很小；③文字尺寸大小变化；④光照条件无法控制，光照不均匀；⑤索书号文字与其他文字共存。

本文利用基于边缘点数量统计的文字图像分割方法。根据图书馆索书号的制定规则，索书号文字图像具有如下特征：①索书号文字采用黑色印刷，②索书号文字的数量至少为 3 个，③索书号字符的水平排列，如图 8-3 所示，④索书号贴在书脊的下半部分。算法主要由以下几个部分组成：①HSI 彩色空间转换；②Canny 算子检测边缘，③索书号文字图像边缘点彩色分割，④文字图像行区域检测，⑤文字图像列区域检测，⑥区域边界调整。



图 8-4　HSI 彩色空间的 I 分量图

1. HSI 彩色空间转换

摄像头拍摄的彩色图片的像素点通常是采用 24 位 RGB 表示。但 R、G 和 B 3 分量之间有很高的相关性，直接利用这些分量常常不能得到所需的分割效果。比较接近人对颜色视觉感知的是色度，饱和度和亮度（HSI）空间。其中 I 表示颜色的明暗程度，H 表示不同的颜色，S 表示颜色的深浅，I 分量与彩色信息无关，H 和 S 分量与人感受彩色的方式紧密相连。

本算法中，只对边缘点进行彩色分割，且只用 S 分量。因此，为了减少算法的耗时，首先只转换 I 分量，如图 8-4 所示。待边缘点检测出来后，再对边缘点进行 S 分量计算。转换关系如下：

$$I = \frac{R+G+B}{3} \tag{8-1}$$

$$S = 1 - \frac{3}{R+G+B}[\min(R,G,B)] \tag{8-2}$$

2. Canny 算子检测

Canny 算子边缘检测先计算 x 和 y 方向的梯度平方和，局部最大的梯度幅值超过阈值的像素点检测为边缘。只保留幅值局部变化最大的点的过程叫非极大值抑制。Canny 算子的边缘检测有如 3 个指标：①对每个边缘点有惟一的响应，得到的边缘为单像素宽；②错误率最低，即

要少将真正的边缘点丢失也要少将非边缘点判为边缘点；③高位置精度，检测的边缘应在真正的边界上。为此，Canny 相应地定义了 3 个准则函数以表达上述指标的约束：①信噪比准则函数；②定位精度准则函数；③单边缘响应准则函数。将 3 个准则相结合可以检测得最佳的边缘。

Canny 算子中有 3 个参数，σ 是高斯函数的分布参数，它控制平滑程度；高阈值（TG）和低阈值（TL）。实验中 σ 设置为 1，$TL=0.4\times TG$。这样在 Canny 算子中只保留了一个参数高阈值（TG）。实际应用中，TG 取值只需满足两个条件：①确保尽可能地检测出索书号文字的边缘点，这粗略限定了 TG 取值的最高上限。②确保假边缘尽可能地少，这限定了 TG 取值的下限。由于文字和背景具有较强的对比度，TG 取值有一个较大的范围，实验中 $TG=0.2$。图 8-4 所示的 Canny 算子边缘检测结果如图 8-5 所示。



图 8-5 Canny 算子检测的边缘

3. 索书号边缘点彩色分割

文字图像有一些独特的特征，文字图像的结构比较复杂，边缘像素点比较多，且分布比较集中。受光照条件和文字颜色退化程度不同的影响，文字部分特征如对比度、亮度等容易受到干扰，但是文字的边缘点分布规律却有较强的抗干扰性能。Canny 算子检测得到的边缘点在 HSI 彩色空间 S 分量上进行索书号文字边缘点分割。由于索书号文字采用黑色印刷，但发生不同程度的褪色，即黑颜色的饱和度 S 有一定的变化，故：

$$BW_Word(i,j)=\begin{cases}1 & S_1\leqslant S\leqslant S_2 \quad and \quad BW(i,j)=1\\0 & \text{其他}\end{cases} \tag{8-3}$$

式中，$BW(i,j)$是 Canny 算子检测的边缘，$BW_Word(i,j)$是文字的边缘点，S_1 和 S_2 表示索书号文字边缘点 S 分量对应的范围。实验中，取 S_1=0，$S_2=0.05$。经边缘点 S 分量分割后的图像如图 8-6 所示，图中的边缘点主要为文字边缘点。



图 8-6 S 分量彩色分割后的边缘

4. 文字图像行区域检测

由于文字图像结构相对比较复杂，其边缘点数量也就比较多，分布也比较集中。因此文字图像区域中行统计平均的边缘点数量比非文字图像区域中的多，则：

$$N_Y^{ave}(word)>N_Y^{ave}(non-word) \tag{8-4}$$

式中，$N_Y^{ave}(word)$ 为文字图像区域的行平均边缘点个数，$N_Y^{ave}(non-word)$ 为非文字图像区域的行平均边缘点个数。

虽然文字图像区域中也存在着字符间的间隔，不能完全保证每一行的边缘点数量都比非文

字图像区域内的行边缘点数量多。但是从概率角度可以认为，文字图像区域内的每一行边缘点数量大于整幅图像边缘点数量的行平均值，而非文字图像区域则相反，即：

$$N_Y(word) > N_Y^{ave} > N_Y(non-word) \tag{8-5}$$

式中 $N_Y(word)$ 是文字图像区域中某一行的边缘点数量，$N_Y(non-word)$ 是非文字图像区域内某一行的边缘点数量，N_Y^{ave} 所示是整幅图像边缘点数量的行平均值。利用（8-5）式检测出可能的文字图像行，图 8-6 的边缘点数量统计如图 8-7 所示。

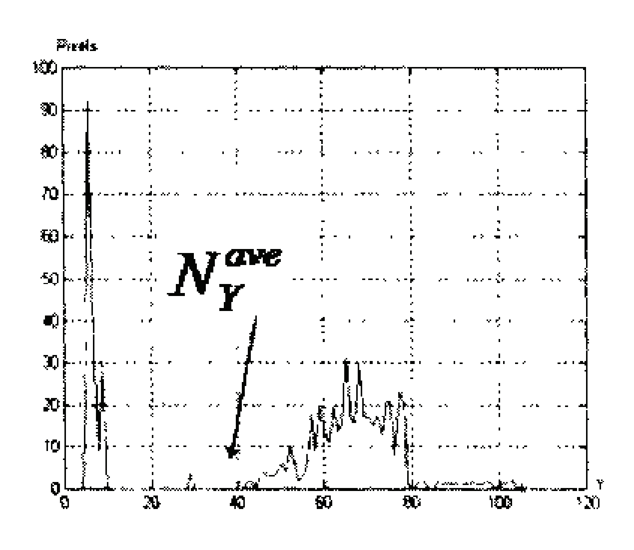


图 8-7　行方向边缘点数量统计

检测出的可能文字图像应先进行合并，形成可能的文字图像区域。受文字图形自身结构繁简程度不同的影响，以及索书号可能由多行字符串组成，而两行字符串之间可能存在间隔，使得文字图像区域中的某些行的边缘点数量低于平均值，设置行方向的间隔阈值 T_Y=10。即如果两可能文字图像行的间距 $D_Y \leq T_Y$，则进行合并。否则视为两个独立的文字图像区域。

经可能文字图像行合并后，图 8-3 所示的索书号文字图像中存在两个索书号图像区域。选择其中包含可能文字图像行最多的区域为索书号文字图像。

5. 文字图像列区域检测

在检测出行区域文字图像中进一步确定。同理：

$$N_X^{ave}(word) > N_X^{ave}(non-word) \tag{8-6}$$

式中，$N_X^{ave}(word)$ 为文字图像区域的列平均边缘点个数，$N_X^{ave}(non-word)$ 为非文字图像区域的列平均边缘点个数。

文字图像区域列边缘点数量也有下述关系：

$$N_X(word) > N_X^{ave} > N_X(non-word) \tag{8-7}$$

式中，$N_X(word)$ 是文字图像区域中某一列的边缘点数量，$N_X(non-word)$ 是非文字图像

区域内某一列的边缘点数量，N_X^{ave}是整幅图像边缘点数量的列平均值。利用（8-7）式检测出可能的文字图像列，图 8-6 的文字图像行区域中边缘点数量列统计如图 8-8 所示。

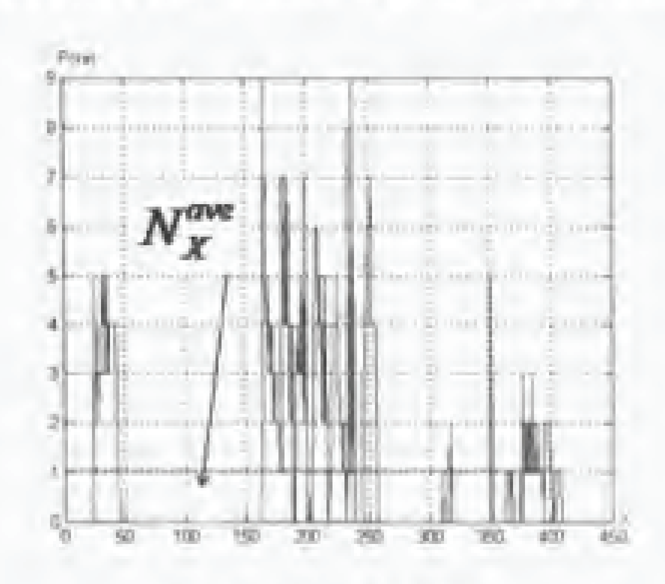


图 8-8　列边缘点数量统计

检测出的可能文字图像列进行合并，形成可能的文字图像列区域。受文字图形自身结构繁简程度不同的影响，以及索书号字符串单个字符之间存在间隔，使得文字图像区域中的某些列的边缘点数量低于平均值，设置行方向的间隔阈值 T_X=20。即如果两可能文字图像列的间距 $D_X \leq T_X$，则进行合并，否则视为两个独立的文字图像列区域。

经可能文字图像列合并后，索书号文字图像中存在两个索书号图像列区域。选择其中包含可能文字图像列最多的区域为索书号文字图像。

6. 边界调整

检测出的文字图像中，由于文字大小写、自身结构等造成字符排列不整齐，导致部分字符的笔划会被漏检，如字符“p”的下半部分。因此，必须调整文字图像的边界。在已经分割出的文字图像区域附近，统计行方向的边缘点数量，若边缘点数量大于 2，则将列并入文字图像区域。同理在列方向进行边界调整。经过列方向和行方向边界调整，形成完整的文字图像区，并提取出文字图像。图 8-3 所示的索书号分割出的索书号图像如图 8-9 所示。



图 8-9　分割出的索书号图像

【例 8-1】索书号文字图像分割

MATLAB 程序代码：

```
%I=imread('24-1.jpg'); 该图像的试验结果读者自行试验
I=imread('22-2.jpg');
tic
[y,x,z]=size(I);
myI=double(I);
```

```
%%%%%%%%%%%% RGB to HSI  %%%%%%%%
HS_I=(myI(:,:,1)+myI(:,:,2)+myI(:,:,3))/3;
t1=toc
tic
%%%% 边缘点数量统计 与 S 分量的纹理分割%%%%%%%%%
S=zeros(y,x);
BW= edge(HS_I,'canny',0.2);%边缘检测，对边缘像素点作统计分析
t2=toc
tic
ES=0;
   for i=1:x
       for j=1:y
           if (BW(j,i)==1)
              S(j,i)=1-min(myI(j,i,:))/HS_I(j,i);
              SR=round(S(j,i)*100)/100;
              ES=[ES SR];
           end
       end
  end
ESx=(0:0.01:1);
BW1=zeros(y,x);
 for i=1:x
       for j=1:y
           % 根据上述的统计直方图设定 S 的阈值
           if ((BW(j,i)==1)&(S(j,i)<=0.06)&(S(j,i)>=0.01))
              BW1(j,i)=1;
           end
       end
  end
t3=toc
tic
%%%%%%%%%%%%%% 边缘像素点数量统计 %%%%%%%%%%%%%%%%%%
 EdgePointsX=sum(BW1); % X方向
 AveragePointsX=mean(EdgePointsX);
 EPx=(1:x);
%%%%%%%%%%%%%% 文字图像区域分割 %%%%%%%%%%%%%%%%%%
              % X Direction
ColumnX=zeros(1,x);
for i=1:x
    if (EdgePointsX(i)>=AveragePointsX)
       ColumnX(i)=1;
    end
end
Nx=sum(ColumnX);
```

```
Posx=zeros(1,Nx); % 挑出边缘点数量大于平均值的位置
k=1;
for i=1:x
    if (EdgePointsX(i)>=AveragePointsX)
        Posx(k)=i;
        k=k+1;
    end
end
gapx=12;              % 间隔小于 10 列的，区域合并
Partx0=zeros(1,Nx);
Partx1=zeros(1,Nx);
k=1;
Partx0(1)=Posx(1);
for i=2:Nx
    d=Posx(i)-Posx(i-1);
    if (d>gapx)
       Partx1(k)= Posx(i-1);
       k=k+1;
       Partx0(k)=Posx(i);
   end
end
Partx1(k)= Posx(Nx);   % 最后一列大于平均值的位置作为，最后一个区域的切分位置
Spanx=zeros(1,k); % 共有 k 个区域
for i=1:k
    Spanx(i)=sum(ColumnX(Partx0(i):Partx1(i)));
end
[mytemp nPartx]=max(Spanx);
word_Xseg0=Partx0(nPartx);
word_Xseg1=Partx1(nPartx);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
BWSeg=BW1(:,word_Xseg0:word_Xseg1);  % Y 方向边缘点统计
 BWT=BWSeg';
 EdgePointsY=sum(BWT);
 AveragePointsY=mean(EdgePointsY);
 EPy=(1:y);

RowY=zeros(1,y);
for i=1:y
    if (EdgePointsY(i)>=AveragePointsY)
        RowY(i)=1;
    end
end
Ny=sum(RowY);
Posy=zeros(1,Ny); % 挑出边缘点数量大于平均值的位置
```

```
k=1;
for i=1:y
    if (EdgePointsY(i)>=AveragePointsY)
        Posy(k)=i;
        k=k+1;
    end
end
gapy=20;            % 间隔小于 30 列的，区域合并
Party0=zeros(1,Ny);
Party1=zeros(1,Ny);
k=1;
Party0(1)=Posy(1);
for i=2:Ny
    d=Posy(i)-Posy(i-1);
    if (d>gapy)
      Party1(k)= Posy(i-1);
      k=k+1;
      Party0(k)=Posy(i);
    end
end
Party1(k)= Posy(Ny);  %  最后一列大于平均值的位置作为，最后一个区域的切分位置
Spany=zeros(1,k); % 共有 k 个区域
for i=1:k
    Spany(i)=sum(RowY(Party0(i):Party1(i)));
end
[mytemp nParty]=max(Spany);
word_Yseg0=Party0(nParty);
word_Yseg1=Party1(nParty);
word=I(word_Yseg0:word_Yseg1,word_Xseg0:word_Xseg1,:);
t4= toc
HS_I=uint8(HS_I);
%%%%%%%%%%%%%%%%%% Display %%%%%%%%%%%%%%%%%%%%
figure(1);
imshow(I);
figure(2);
imshow(HS_I);
figure(3);
imshow(BW);
figure(4);
hist(ES,ESx);grid
title('S Histogram of the Edge Pixels');
figure(5);
imshow(BW1);
title('The most is the Character Edge pixels in the Histogram');
```

```
figure(6);
plot(EPx,EdgePointsX);hold on
plot(EPx,AveragePointsX,'r');grid
title('X Direction Edge Pixels Statistics');
figure(7);
plot(EPy,EdgePointsY);hold on
plot(EPy,AveragePointsY,'r');grid
title('Y Direction Edge Pixels Statistics');
figure(8);
imshow(word);
title('Extraction Character');
```

程序结果如图 8-9 所示。

8.2.3 文字图像二值化

由于 CCD 摄像时的光照不均匀，褪色程度差别较大等影响，不可能对所有索书号图像设定一固定阈值进行二值化。此处，采用前面已经讨论分析过最佳阈值二值化方法，对分割出的灰度索书号图像自动确定阈值，根据各自索书号图像灰度分布特点进行动态二值化。图 8-3 索书号的二值化结果如图 8-10 所示。

F743/13

图 8-10　索书号的二值化

8.2.4 单个字符的切分

目前，OCR（Optical Character Reader）文字识别技术已经比较成熟，一般质量的印刷体单字识别的正确率已经高达 99%以上。但是，粘连字符识别错误率还比较高，一般认为主要原因是粘连字符的错误切分而导致字符的严重失真变形，无法实现正确的识别，因此粘连字符的切分成为提高识别率的关键技术。现有的切分方法主要有：①基于图像分析的直接切分法，通过图像分析寻找字符之间较为合理的切分点，这种切分的错误率比较高；②基于识别的切分方法，先通过图像分析，确定几个可能的切分点，借助识别结果，选择合理的切分点。这种切分方法的识别比较高，但是多次识别，比较耗时，速度慢。

分析污点、褪色、光照不均匀和分辨率过低对索书号字符粘连的影响，表明字符粘连处文字笔划可能发生了较大的变形，错误粘连切分固然改变了字符形状，但是合理的切分也不能完全纠正笔划的变形。为了提高切分速度，提出了采用上下轮廓凹凸特征近似检测单个字符的宽度，在字符宽度的约束下，根据轮廓凹凸特征，直接建立切分路径。并利用切分处笔划宽度特征，对切分处的变形笔划进行了恢复，有效抑制噪声干扰而产生的字符笔划变形，提高索书号文字识别的正确率。

1. 字符粘连的主要原因

文档图像本身是二值图像，其粘连的主要原因是低劣的印刷质量和图像分辨率较低而引起的，字符笔划变形主要字符粘连和粘连笔划的切分引起。因此，准确的切分路径就能纠正字符变形，保证切分后字符识别较高的正确率。与文档图像相比，索书号字符图像具有以下特点：①字符间隔比文档中单词的字符间隔大；②索书号的字符基本保持相同高度；发生重叠粘连和交错粘连的概率很小。但是，从书架上的图书侧翼分割出的索书号字符图像是彩色图像或灰度图像，字符识别之前需要二值化处理。而且索书号字符图像严重受下列因素的干扰：①字符串附近存在的污点；②字符和纸张的不均匀褪色；③光照不均匀。这些因素的存在，使字符图像在二值化处理时，不可避免地会将部分背景区域错误地分为字符笔划，致使字符笔划发生变形而导致粘连，即字符粘连处也是笔划变形程度相当严重的地方。因此，准确的切分后还必须纠正粘连处的笔划变形。另一方面，字符变形导致图像分析容易出错，切分点的选择和切分路径的建立比较困难。

2. 字符串凹凸轮廓定义及检测

图书馆藏书的索书号文字图像经过文字提取分割后得到的字符串，经阈值分割后的二值化图像如图 8-11 所示。定义字符串上轮廓为字符图像顶部到字符最高点得距离：

$$T_p(i) = ET(i), \qquad i = 1, 2, \cdots, W \tag{8-8}$$

式中 $ET(i)$为 I 列图像上的字符最高点，W 为字符图像的宽度。如果所在列无字符，则上轮廓为字符图像的高度 0。同理，定义字符串下轮廓为字符图像底部到字符最低点的距离：

$$B_p(i) = H - EB(i), \qquad i = 1, 2, \cdots, W \tag{8-9}$$

式中 $EB(i)$为 I 列图像上的字符最低点。如果所在列无字符，则下轮廓为字符图像的高度 H。



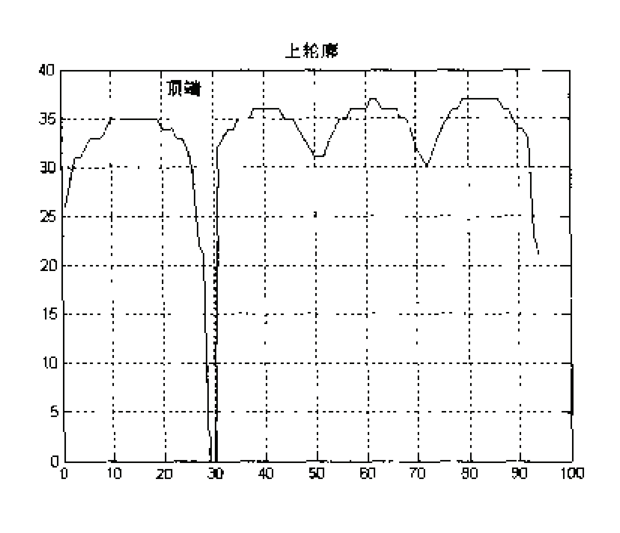
图 8-11　二值化的索书号字

根据上述定义检测到的索书号字符的上下轮廓如图 8-12(a)和 8-12(b)所示。在字符间的间隔处，则在上轮廓存在凹结构，在下轮廓存在凸结构。上轮廓的离散微分为：

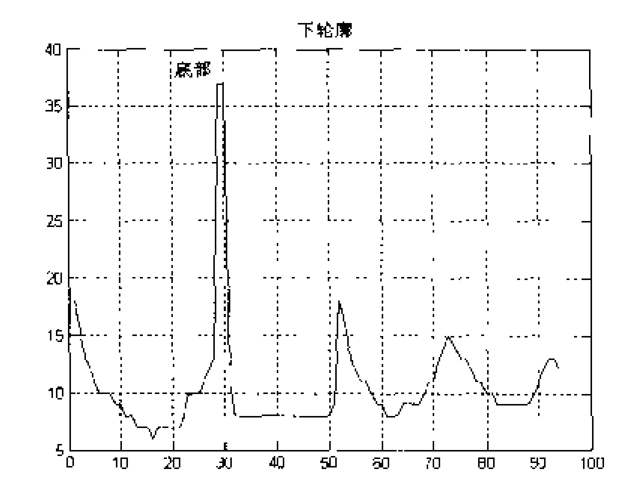
$$TD_p(i) = T_p(i+1) - T_p(i), \qquad i = 1, 2, \cdots, W-1 \tag{8-10}$$

下轮廓的离散微分为：

$$BD_p(i) = B_p(i+1) - B_p(i), \qquad i = 1, 2, \cdots, W-1 \tag{8-11}$$



(a) 字符串上轮廓曲线



(b) 字符串下轮廓曲线

图 8-12

其中$1\leqslant j\leqslant W-2$，$1\leqslant k\leqslant W-j-1$。凹结构的位置为 *CT*:

$$CT=\begin{cases} j & k=1 \\ j+[\frac{k}{2}] & k\geqslant 2 \end{cases} \tag{8-12}$$

凸轮廓定义：$BD_p(j)>0$，而$BD_p(j+k)<0$，若$k\geqslant 2$，则$\forall i\in(j,j+k),BD_p(i)=0$，

其中$1\leqslant j\leqslant W-2$，$1\leqslant k\leqslant W-j-1$。凸结构的位置为 *CB*:

$$CB=\begin{cases} j & k=1 \\ j+[\frac{k}{2}] & k\geqslant 2 \end{cases} \tag{8-13}$$

在凹凸轮廓实际检测时，由于字符图像受噪声影响，字符的部分笔划边缘不光滑，会检测到一些仅有 1、2 个像素深度或高度的假凹轮廓或假凸轮廓。为了有效抑制笔划边缘不光滑的干扰影响，在检测过程中，增加了一个约束条件，即凹轮廓的深度和凸轮廓的高度必须大于等于 3 个像素。

3. 字符高度和宽度的近似检测

根据上轮廓和下轮廓的定义，可以检测到各列字符图像的高度 $H(i)$:

$$H(i)=EB(i)-ET(i)=H-B_p(i)+T_p(i) \tag{8-14}$$

取字符串的最大高度为字符的近似高度 H_W:

$$H_W=\max_{i=1}^{W}[H(i)] \tag{8-15}$$

在字符串图像的上下轮廓曲线表明：非粘连字符的间隔处存在鲜明的凹凸结构；绝大部分

粘连字符在间隔处也会同样存在凹凸结构，小部分水平笔划粘连的字符至少也会存在凹结构或凸结构。因此，可以采用凹凸结构的间距近似检测单个字符的宽度 G_W。

由于凹凸结构受字符自身形状和粘连情况的影响，如 I、I 和 l（小写 L）和 1 等字符相对较窄，而 M、N、V 和 W 等字符本身包含凹凸结构，从这些字符检测到的间距会偏小，而水平笔划的字符粘连会使检测到的间距偏大，所以取凹结构间距和凸结构间距的中间值作为单个字符的近似宽度。设 n 个凹结构的位置集合为{$CT(1)$，$CT(2)$，…，$CT(n)$}，凸结构位置集合为{$CB(1)$，$CB(2)$，…，$CB(m)$}，则凹结构和凸结构的间距分别为

$$D_{CT}(i) = CT(i) - CT(i-1),\quad i = 2,3,\cdots,n \tag{8-16}$$

$$D_{CB}(i) = CB(i) - CB(i-1),\quad i = 2,3,\cdots,m \tag{8-17}$$

为了将第一个凹结构和凸结构的间距包括在内，定义第一个凹结构和凸结构的间距为：

$$D_{CT}(1) = CT(1) - L_C \tag{8-18}$$

$$D_{CB}(1) = CB(1) - L_C \tag{8-19}$$

其中 L_C 为第一列字符串图像列。

单个字符的宽度估计值为：

$$G_W = med\{D_{CT}(1),\cdots,D_{CT}(n),\cdots,D_{CB}(1),\cdots,D_{CB}(m) \tag{8-20}$$

字符高度的检测比其宽度检测更加准确可靠，印刷体的字符高度和宽度之间满足如下关系：

$$G_W \cong 0.7H_W \tag{8-21}$$

因此，如果当字符因干扰严重时，根据上述关系，建立字符宽度检测的约束条件：

$$0.5H_W \leqslant G_W \leqslant 0.8H_W \tag{8-22}$$

即当用凹凸结构轮廓检测的字符宽度不能满足（8-22）式的约束关系时，用（8-21）式计算字符宽度。

4. 粘连字符的切分

索书号字符粘连的类型主要是简单粘连。上下轮廓的凹凸结构位置作为可能的切分列，在如下约束条件的作用下进行切分：

同一索书号的字符尺寸大小相同，则每个字符的宽度 L_W 应该在一定范围内，即：

$$0.6G_W \leqslant L_W \leqslant 1.5G_W \tag{8-23}$$

在上述约束条件的作用下，切分算法如下所示。切分的步骤如下：

第 1 步，为非粘连字符的切分。检测上轮廓的凹结构，若第 i 个凹结构的 $T_p(CT(i)) = 0$，则 i 凹结构为非粘连字符的间隔，切分后所得的字符区域为{$P_1,P_2,\cdots,P_N$}。

第 2 步，在上述切分的基础进行粘连字符的切分。算法如图 8-13 所示。

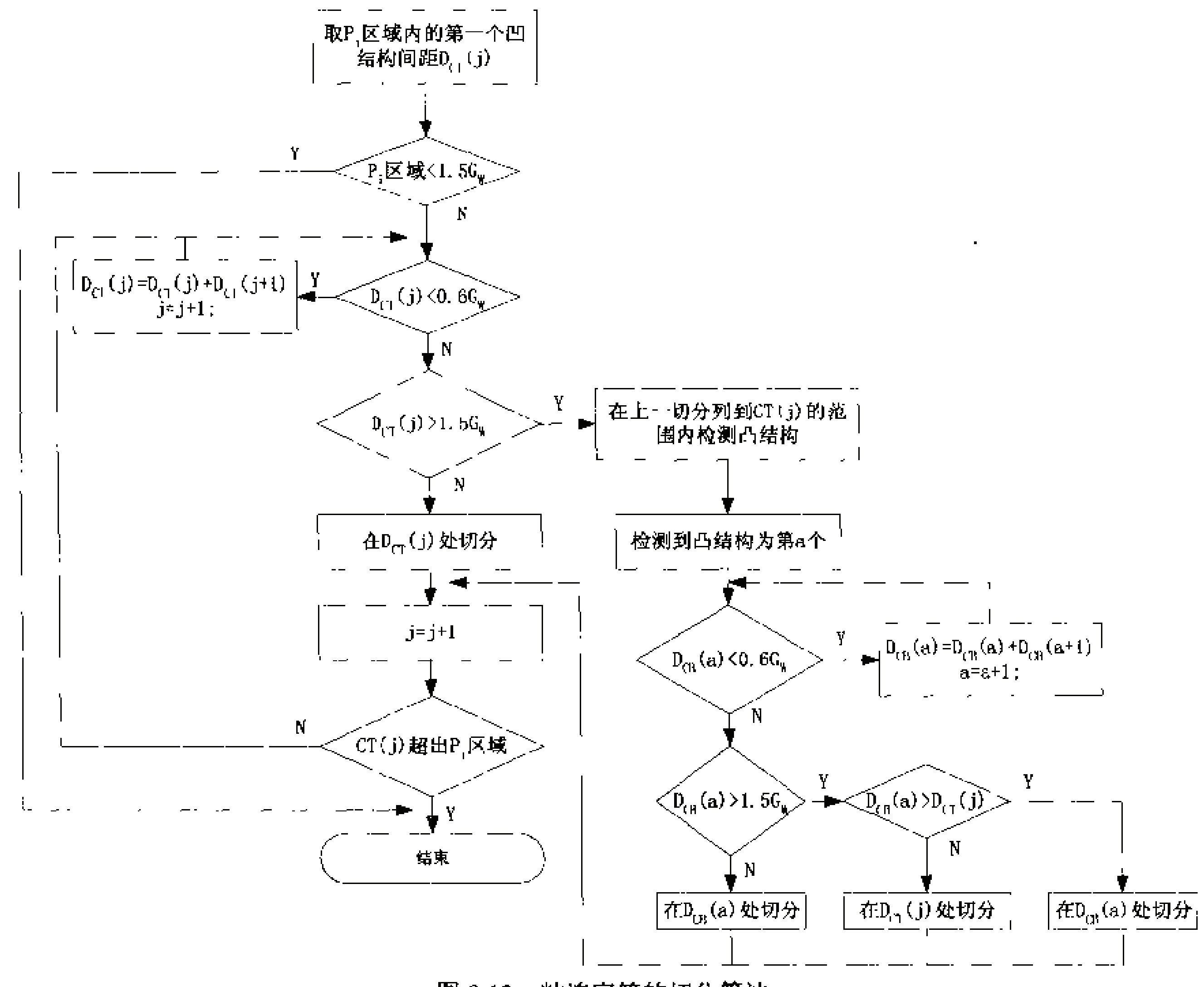


图 8-13 粘连字符的切分算法

5. 字符变形笔划修复

粘连字符的切分是对字符变形的纠正，但并不是正确的切分就能完全修复字符笔划的变形。如图 8-14 所示，正确的切分后，字符“3”的变形。由于索书号字符串的字符笔划宽度接近相等，粘连笔划切分后同样应该满足该条件。设切分后，切分路径经过的左边笔划的宽度为 S_L，右边的笔划宽度为 S_R，则笔划宽度之比 R_W:

$$R_W = \min\{\frac{S_L}{S_R}, \frac{S_R}{S_L}\}$$

若 $R_W \leqslant 0.5$，则消除笔划宽度较小的笔划。有时切分，会使上式中的 S_R 或 S_L 为 0，无法进行计算分析，因此，当其中任意一直为 0 时，则令 $R_W=1$。经过笔划修复后，字符“3”可以回复笔划形状，如图 8-15 所示。

G 2 0 3

图 8-14 粘连字符的切分结果

3

图 8-15　变形笔划的修复

【例 8-2】粘连字符切分

MATLAB 程序代码：

```
I=imread('19-3-02.jpg');
%I=imread('49-3-BW01.jpg');
I=im2bw(I,0.43);
[y x]=size(I);
Top=zeros(1,x);          % 顶端轮廓检测
for i=1:x
      j=1;
   while ((I(j,i)==1)&&(j<y))
      j=j+1;
   end
   Top(i)=y-j;
end
Bottom=zeros(1,x);       % 底部轮廓检测
for i=1:x
      j=y;
   while ((I(j,i)==1)&&(j>1))
      j=j-1;
   end
   Bottom(i)=y-j;
end
Height=Top-Bottom;
WordHeight=max(Height);    % 文字高度
%=== 轮廓线的凹检测 ===%
TopD=zeros(1,x-1);
Concave=1;                   % 记录凹轮廓处的位置，1 表示默认起始列为第一个 Concave
Deep=0;                       % 下降值
DeepH=0;                      % 上升值
DeepT=5;                      % 凹轮廓的深度阈值
Sign=0;
for i=1:x-1
   TopD(i)=Top(i+1)-Top(i);
end
for i=1:x-2
   if (TopD(i)<0)  % 判断是否为凹轮廓
      Sign=1;   % 置标志位
      DeepH=0;
      Deep=Deep+TopD(i);
      tempX=i+1;  % 下一列可能为切分的 Concave， 最接近于左端
```

```
    end
    if ((Sign==1)&&(TopD(i)>0))
        if (abs(Deep)>=DeepT)
            DeepH=DeepH+TopD(i);
            if (abs(DeepH)>=DeepT)
                Concave=[Concave tempX];
                Sign=0;     % 确认为凹后，复位标志位
                DeepH=0;
            end
        else
            Sign=0;     % 确认为凹后，复位标志位
            Deep=0;
        end
    end
end
%=== 轮廓线的凸检测 ===%
BottomD=zeros(1,x-1);
Convex=1;
Asend=0;                        % 上升值
Desend=0;                       % 下降值
ConvexT=3;                      % 凸程度阈值
Sign=0;
for i=1:x-1
    BottomD(i)=Bottom(i+1)-Bottom(i);
end
for i=1:x-2
    if (BottomD(i)>0)
        Sign=1;
        Desend=0;
        Asend=Asend+BottomD(i);
        tempX=i+1;          % 最接近于左端
    end
    if((Sign==1)&&(BottomD(i)<0))
        if (abs(Asend)>=ConvexT)
            Desend=Desend+BottomD(i);
            if (abs(Desend)>=ConvexT)
                Convex=[Convex tempX];
                Sign=0;             % 复位
                Desend=0;
            end
        else
        Sign=0;             % 复位
        Asend=0;
        end
```

```
    end
end
%=== 切分 ===%
[mytemp n]=size(Concave);   % 注意 Concave 的第一个数值无效
StrokeT=5;                  % 笔划宽度阈值
GapT=8;
W=zeros(1,n);
for i=1:n-1
    W(i)=Concave(i+1)-Concave(i);
end
W(n)=x-Concave(n);
Width=median(W);            % 近似的字符宽度
    PXR1=1;                 % 记录第一次切分位置
    PXR2=1;                 % 记录第二次切分位置
    Mark=0;           % 记录黑白转换的次数
    %CrossSign=0;     % 交错粘连的标志
    Black=zeros(1,x);         % 统计笔划像素点
    BP=zeros(1,x);
    SegSoke=zeros(3,x);       % 切分点处的笔划宽度
    RH=zeros(1,x);           % 切分后的高度比
    RW=zeros(1,x);           % 切分后的宽度比
    Score=zeros(1,x);        % 特征值的总得分
    XGood=1;                % X 切分位置
    SegY=1;           % 记录第一次切分的 Y 深度
for k=2:n
    WordH=max(Height(Concave(k-1):Concave(k)));
    WordW=Concave(k)-Concave(k-1);
    if ((WordW>=0.5*Width)&&(WordW<=1.5*Width))
        % 选定切分的区域
        PX1=Concave(k);
        PX2=PX1;
        while  ((TopD(PX2)==0)&&(PX2<x))
               PX2=PX2+1;                  % 凹右边的列位置
        end
       i=fix((PX1+PX2)/2);
       if  (Top(i)==1)                 % 无粘连
           PXR1=[PXR1  i];
           PXR2=[PXR2  i];
       else
           j=y+1-Top(i);                % PY 为实际的 y 坐标值，此处已为黑色像素点
           Mark=0;
           while((j<y)&&(Mark<2))
               if  (I(j,i)==0)
                   Black(i)=Black(i)+1;  % 记录黑色像素点数
```

```
                Si=i;
                while ((Si>1)&&(I(j,Si)==0))   % 左笔划宽度
                        Si=Si-1;
                        SegSoke(1,i)=SegSoke(1,i)+1;
                end
                Si=i;
                while ((Si<x)&&(I(j,Si)==0))   % 右笔划宽度
                       Si=Si+1;
                       SegSoke(2,i)=SegSoke(2,i)+1;
                 end
            end
             Mark=Mark+abs(I(j+1,i)-I(j,i));  % 检测是否通过笔划
             j=j+1;
          end
          SegY=[SegY j-1];                        % 第一次切分截止处
          if (j==38)
              PXR1=[PXR1 i];
              PXR2=[PXR2 i];
          else                                   % 单点粘连
              SLi=i;
              while ((SLi>1)&&(I(j-1,SLi)==1))         % 选定区域左边界
                    SLi=SLi-1;
              end
              SRi=i;
              while ((SRi<x)&&(I(j-1,SRi)==1))         % 选定区域右边界
                    SRi=SRi+1;
              end
              [Mytemp PX2]=max(Bottom(SLi:SRi));
              PXR2=[PXR2 PX2+SLi-1];
              PXR1=[PXR1 i];
          end
      end
    else if (WordW>1.5*Width)
            PX=fix((Concave(k)+Concave(k-1))/2);  % 避免水平“横”的粘连
            k=k-1;
        end
                    %  如果宽度过小，则不切分
  end
end
%==== Segment ===%
for i=2:n
   WI=I(:,fix((PXR1(i-1)+PXR2(i-1))/2):fix((PXR1(i)+PXR2(i))/2));
   figure(10+i);imshow(WI);
end
```

```
WI=I(:,fix((PXR1(i)+PXR2(i))/2):x);
figure(10+i+1);imshow(WI);
%=== 图像显示 ===%
px=(1:x);                    % X轴坐标
figure(1);
imshow(I);
figure(2);
plot(Top);hold on
plot(px,y,'red');grid
title('上轮廓');
figure(3);
plot(Bottom);hold on
plot(px,y,'red');grid
title('下轮廓');
figure(4);
plot(Height);grid
```

程序结果如图 8-14 所示。

8.2.5　文字识别

目前为止，已经有成熟的 OCR 商业软件，或者其他形式的产品。读者可以参考扫描仪自带的尚书 OCR 文字识别软件。但有时在实际应用中，要根据实际需要开发精简的识别算法。本节主要介绍索书号中的数字图像文字识别，为读者提供一个具体范例。

采用数字字符轮廓结构特征和统计特征相结合的方法，并从中选出稳定的局部特征，利用结构语句识别的方法进行数字的识别，能够实现不同字体索书号多种字体数字的准确识别，同时还提高了识别的速度。

1. 字符轮廓定义

由于受噪声和随机污点的干扰，以及二值化和粘连字符处理会引起字符的变形。为了尽量减少这种变形对信息特征的干扰，或者从变形的字符中提取可靠的特征信息，将字符的整体轮廓分解为顶部、底部、左侧和右侧 4 个方向的轮廓特征来描述，使得当其中某部位的笔划发生变形时，不会改变或者减少对其他部位特征的影响。

左侧轮廓（$LP(k)$，$k=1,2,\cdots,M$）定义为字符最左侧边界像素点的水平方向坐标值。

$$LP(i)=\min\{x \mid P(x,y)\in C,\quad y=i\}\quad i=1,2\cdots M \tag{8-24}$$

式中 $P(x,y)$表示图像中坐标为(x,y)的像素点，C 表示字符像素点的集合。同理，右侧轮廓（$RP(k)$，$k=1,2,\cdots,M$）定义为字符最右侧边界像素点的水平方向坐标值。

$$RP(i)=\max\{x \mid P(x,y)\in C,\quad y=i\}\quad i=1,2\cdots,M \tag{8-25}$$

相应地，顶部轮廓（$TP(k)$，$k=1,2,\cdots,N$）定义为字符最高边界像素点的垂直方向坐标值。底部轮廓（$BP(k)$，$k=1,2,\cdots,N$）定义为字符最低边界像素点的垂直方向坐标值。轮廓定义如图 8-16 所示。

$$TP(j)=\min\{y \mid P(x,y)\in C,\quad x=j\}\qquad j=1,2\cdots,N \tag{8-26}$$

$$BP(j)=\max\{y \mid P(x,y)\in C,\quad x=j\}\qquad j=1,2\cdots,N \tag{8-27}$$

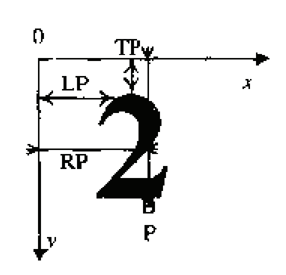


图 8-16 字符轮廓定义示意图

为了描述轮廓的变化特征，定义 4 个方向轮廓的一阶微分：

$$\begin{aligned}
LPD &= LP(i+1)-LP(i)\\
RPD &= RP(i+1)-RP(i)\\
TPD &= TP(j+1)-TP(j)\\
BPD &= BP(j+1)-BP(j)
\end{aligned} \tag{8-28}$$

式中 $i=1,2,\cdots,M\text{-}1$，$j=1,2,\cdots,N\text{-}1$。

2. 结构基元

利用（8-28）式的轮廓一阶微分变化趋势，定义构成字符轮廓的基本基元。基本基元共有 5 个分别为：左斜（*L*）、右斜（*R*）、竖直（*V*）、圆弧（*C*）和突变（*P*）。以左侧轮廓为例，定义上述基本基元：

（1）竖直

定义：假设 *SL*，*SV* 和 *SR* 分别表示某侧轮廓一阶微分值大于零，等于零和小于零的个数，若 *SR*=0，*SL*=0，则为结构 *V*，如图 8-17 (a)所示。

（2）左斜

定义：假设 *SL*，*SV* 和 *SR* 分别表示某侧轮廓一阶微分值大于零，等于零和小于零的个数，若 *SR*=0，*SL* 大阈值 *LT*，则为结构 *L*，如图 8-17 (b)所示。

（3）右斜

定义：假设 *SL*，*SV* 和 *SR* 分别表示某侧轮廓一阶微分值大于零，等于零和小于零的个数，若 *SL*=0，*SR* 大阈值 *RT*，则为结构 *R*，如图 8-17 (c)所示。

（4）圆弧

定义：假设 *SL*，*SV* 和 *SR* 分别表示某侧轮廓一阶微分值大于零，等于零和小于零的个数，若 *SR* 大于阈值 *RT*，*SL* 大阈值 *LT*，则为结构 *C*，如图 8-17 (d)所示。需要指出的是，圆弧示意图只是一种抽象，它表示结构中包含了上升和下降的两种趋势，而不仅仅只是图 8-17 (d)所示的具体形状。

（5）突变

连续的字符轮廓，其一阶微分值的变化量比较小，而当字符轮廓不连续时，其一阶微分值相对较大。因此，定义：当轮廓的一阶微分值超过阈值 *PT* 时则字符轮廓有突变，即为结构 *P*，

如图 8-17 (e)所示。

基元结构示意图如图 8-17 所示。

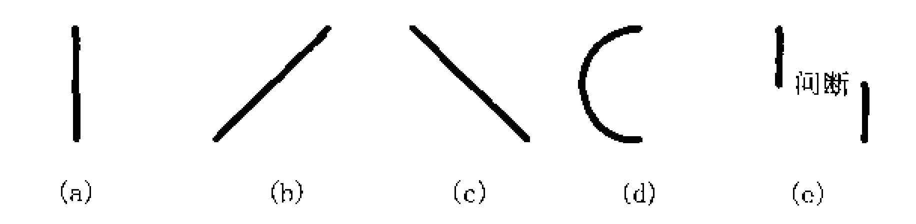


图 8-17　基元结构示意图

3. 基元的检测

根据上述定义，考虑实际应用中存在的干扰，基元的检测规则如下：

假设 $PD(k)$表示某侧轮廓的一阶微分，$k=1,2,\cdots,K$，SL，SV 和 SR 分别为检测到的 $PD(k)$ 大于零，等于零和小于零的个数，PT、RT 和 LT 为正整数，则：

（1）若 $|PD(k)| \geq PT$，则在 k 处检测到结构突变(P)；

若 $SL < LT$，$SR < RT$，则检测到结构为竖直(V)；

若 $SL > LT$，$SR < RT$，则检测到结构为左斜(L)；

若 $SL < LT$，$SR > RT$，则检测到结构为右斜(R)；

若 $SL > LT$，$SR > RT$，则检测到结构为圆弧(C)。

（2）由于字符轮廓突变处，表示字符轮廓不连续，则突变前后的轮廓特征必须分别检测。即若 k_1 处检测到 P，则在$[1,k_1-1]$的字符轮廓范围内统计 SL，SV 和 SR 独立进行结构基元检测。若在 k_2 处又检测到 P，则在$[k_1+1,k_2-1]$范围内进行基元检测，依此类推。

（3）由于字符轮廓基元的形成需要一定数（T）轮廓像素点，即只有当 $SL+SV+SR \geq T$ 时，才能进行基元检测，否则不进行基元检测。例如，当 $SL+SV+SR=2$ 时，其形成的基元结构是不稳定。

（4）检测到突变结构 P 的有效范围在 $x \in [ST, N-ST+1]$，$y \in [ST, M-ST+1]$，其中 ST 表示字符笔划的宽度。这主要是为了避免干扰严重情况下，轮廓边缘光滑处理不够理想时，可能检测到的假突变基元。

4. 轮廓的统计特征

采用上述的结构基元还不足以准确识别残缺和完整的数字，引入轮廓的统计特征。

（1）水平方向的最大字符宽度 W_{max}：

$$W_{max} = \max_k \{RP(k) - LP(k)\} \tag{8-29}$$

该特征主要用于识别数字 1。当 $W_{max} \leqslant \frac{H}{2}$，即为数字 1，$H=M$。

（2）垂直方向的笔划数

该特征主要用于识别数字 0 和 8。因为 0 和 8 的轮廓结构特征及其相似，所以借助于垂直方向的笔划数加以区分。受数字底部残缺的影响，8 在垂直方向的最大笔划数也可能为 2。采用检测到笔划数为 2 时垂直方向的最小值来代替。假设 j 列上像素点 $P(j,i-1)$，检测到垂直方向的笔划数为 1，在 $P(j,i)$检测到了第二个笔划，则 $S_2=i$，表示检测到第二个笔划的像素点位置。当 $S_2<M-ST$ 时，检测到的字符为 8，否则为 0。

5. 数字字符的识别算法

将数字字符的顶部、左右两侧的局部轮廓结构特征和轮廓统计特征组合成特征向量，用以描述 10 个数字。根据特征向量，采用用结构语句识别算法识别底部残缺的和完整的数字字符。由于底部特征丢失，会改变左右两侧的部分结构特征，但不会影响顶部特征，因此特征描述和机构匹配识别都从顶部轮廓特征开始。局部轮廓结构特征和统计特征描述数字：

0：$TS=C$，$\mathrm{Size}(LS)=\mathrm{Size}(RS)=1$；$S_2<M-ST$。

式中，$Size()$ 表示结构集合中有几个结构元素。

1：$W_{max}\leqslant H/2$。

2：$TS=C$，$LS(1)\neq C$，$LS(Ln-1)=P$，$LS(Ln)=L$。

式中，Ln 表示左侧轮廓的结构元素个数。

3：$TS=C$，$LS(1)\neq C$，$P\in LS$；或 $TS=V$，$RS=C$。

4：$TS(1)=L$，$P\in TS$，$RS=V$。

5：$TS=V$，$P\in RS$。

6：$TS=C$，$P\in RS$，$\mathrm{Size}(LS)=1$；或 $TS(1)=L$，$V\notin RS$。

7：$TS=V$，$P\in LS$，$\mathrm{Size}(RS)=1$。

8：$TS=C$，$\mathrm{Size}(LS)=\mathrm{Size}(RS)=1$；$S_2>M-ST$。

9：$TS=C$，$LS(1)=C$，$LS(2)=P$。

【实例 8-3】文字识别

MATLAB 程序代码：

（1）主程序

```
I0=imread('9.jpg');% 必须为二值图像
I=im2bw(I0,0.4);
[y0 x0]=size(I);
Range=sum((~I)');
Hy=0;
for j=1:y0
    if (Range(j)>=1)
        Hy=Hy+1;
    end
```

```
end
RangeX=sum((~I));

Wx=0;
for i=1:x0
    if (RangeX(i)>=1)
        Wx=Wx+1;
    end
end
Amp=24/Hy;              % 将文字图像归一化到 24 像素点的高度。
I=imresize(I,Amp);
[y,x]=size(I);
%I=bwmorph(~I,'skel',Inf);
%I=~I;
tic
%====== 基本结构 =======%
% 第一类: 竖 (V) ; 左斜 (L) ; 右斜 (R) ; 突变 (P)
% 第二类: 左半圆弧 (C) ; 右半圆弧 (Q)
% 的三类: 结构待定 (T) ;
%=====================================%
Left=zeros(1,y);            % 左端轮廓检测
for j=1:y
        i=1;
    while ((i<=x)&&(I(j,i)==1))
        i=i+1;
    end
    if (i<=x)
        Left(j)=i;
    end
end
for j=1:y-1
    LeftD(j)=Left(j+1)-Left(j);
end
%========== 结构特征提取 =============%
j=1;
while ((Left(j)<1)&&(j<y))
        j=j+1;
end
Y1=j;
j=y;
while ((Left(j)<1)&&(j>1))
        j=j-1;
end
Y2=j-1;         % 去掉急剧变化的两端
```

```
%============== 右边 ================%
Right=zeros(1,y);          % 左端轮廓检测
for j=1:y
      i=x;
   while ((i>=1)&&(I(j,i)==1))
       i=i-1;
   end
   if (i>=1)
       Right(j)=i;
   end
end
for j=1:y-1
   RightD(j)=Right(j+1)-Right(j);
end
%====================================%
Top=zeros(1,x);          % 顶端轮廓检测
for i=1:x
       j=1;
   while ((j<=y)&&(I(j,i)==1))
       j=j+1;
   end
   if (j<=y)
       Top(i)=j;
   end
end
for i=1:x-1
   TopD(i)=Top(i+1)-Top(i);
end
%=============================%
i=1;
while ((Top(i)<1)&&(i<x))
       i=i+1;
end
X1=i;
i=x;
while ((Top(i)<1)&&(i>1))
       i=i-1;
end
X2=i-1;       % 去掉急剧变化的两端
%===================================%
Bottom=zeros(1,x);       % 底部轮廓检测
for i=1:x
       j=y;
   while ((j>=1)&&(I(j,i)==1))
```

```
        j=j-1;
    end
    if (j>=1)
        Bottom(i)=j;
    end
end
for i=1:x-1
    BottomD(i)=Bottom(i+1)-Bottom(i);
end
%========== 数字 1 的宽度特征 =========%
Width=zeros(1,y);
for j=1:y
    Width(j)=Right(j)-Left(j);
end
W=max(Width);
Po=0; % 用于检测笔划
Ne=0;
NS=0;%笔划数
for i=X1+4:X2-4     % 消除笔画宽度区域的不稳定范围
    for j=1:y-1
        if ((I(j+1,i)-I(j,i))>0) % 由黑到白
            Po=Po+1;
            if ((Po>=2)&&(j<=fix(0.7*y)))
                Po=3;
            end
        else if  ((I(j+1,i)-I(j,i))<0) % 由白到黑
                Ne=Ne+1;
                if ((Ne>=2)&&(j<=fix(0.7*y)))
                    Ne=3;
                end
            end
        end
    end
    NS=[NS max(Po,Ne)];
    Po=0;
    Ne=0;
 end
 Comp=max(NS);
%========== 轮廓结构特征提取 ==========%
if (min(W,Wx)>10)
    StrokeT=StrDetect01(TopD,X1,X2,3,6); % 顶部基本结构检测
    StrokeL=StrDetect01(LeftD,Y1,Y2,3,5); % 左边基本结构检测
    StrokeR=StrDetect01(RightD,Y1,Y2,3,5); % 右边基本结构检测
    StrokeB=StrDetect01(BottomD,X1,X2,3,6); % 底部基本结构检测
```

```
%========== 识别 ==========%
   Digit=Recognition(StrokeT,StrokeL,StrokeR,StrokeB,Comp)
else
   Digit='1'
end
t=toc
%======= 显示 ======%
px=(1:x);
py=(1:y);
figure(1);
imshow(I);
figure(2);
plot(Left);grid
title('左轮廓');
figure(3);
plot(Top);grid
title('上轮廓');
figure(4);
plot(Right);grid
title('右轮廓');
figure(5);
plot(Width);grid
title('宽度');
```

（2）结构基元检测程序 StrDetect01

```
unction [Stroke]= StrDetect01(LeftD,Y1,Y2,ST,PT)
% ST 为结构阈值，为了指定高度和宽度结构变化的不同
SL=0;
SR=0;
SV=0;
Count=0;
%PT=5;    % 突变的阈值
Str='T'; % T 表示结构未定，Str 用于保存当前的基本结构
Stroke='T'; % 用于保存基本结构
Range=Y2-Y1+1; % 字符的宽度或者高度
for j=Y1:Y2
    Count=Count+1;

    if (abs(LeftD(j))<PT)
          if (LeftD(j)<0)
                SL=SL+1;
             else if (LeftD(j)>0)
                SR=SR+1;
             else
                SV=SV+1;
```

```
                end
            end
    else        % 检测到突变的决策
        if ((Count>=fix(Range/4)+1))    % 设定字符轮廓可能发生的突变范围
            if ((SL>=3)&&(SR>=3))
                Str='C';
            else if ((SV>=2*(SL+SR))&&((max(SL,SR)<3)||(min(SL,SR)<2)))
                Str='V';
                else if (((SL>SR)&&((SL>=0.5*SV)&&((SR<=1)||(SL>(SR+SV)))))
                    Str='L';
                    else if (((SR>SL)&&((SR>=0.5*SV)&&((SL<=1)||(SL>(SR+SV)))))
                        Str='R';
                        else if (max(SL,SR)>=3)&&(min(SL,SR)>=2)
                            Str='C';
                        end
                    end
                end
            end
        end
        Stroke=[Stroke Str];
    end
        if ((j>=2+Y1)&&((j<=Y2-2)))
            Stroke=[Stroke 'P'];
        end
        SL=0;
        SR=0;
        SV=0;
        Count=0;
        Str='T';
    end
end
%========= 提取结构 ===============%
if (Count>=fix(Range/4)+1) % 发生突变后，剩余部分可能无法形成字符结构
if ((SL>=ST)&&(SR>=ST))
   Str='C';
else if ((SV>=2*(SL+SR))&&((max(SL,SR)<3)||(min(SL,SR)<2)))
      Str='V';
   else if (((SL>SR)&&((SL>=0.5*SV)&&((SR<=2)||(SL>=(SR+SV)))))
         Str='L';
      else if (((SR>SL)&&((SR>=0.5*SV)&&((SL<=2)||(SL>=(SR+SV)))))
            Str='R';
         else if (max(SL,SR)>=3)&&(min(SL,SR)>=2)
               Str='C';
            end
```

```
            end
         end
      end
   end
Stroke=[Stroke Str];
end
```

（3）结构识别 Recognition

```
function [Numeral]=Recognition (StrokeTop,StrokeLeft,StrokeRight, Stroke
Bottom, Comp)
% 采用四边的轮廓结构特征和笔划统计（仅针对 0 和 8）识别残缺数字
% Comp 是用于识别 0 和 8 的底部补充信息
StrT='T';
StrL='T';
StrR='T';
StrB='T';
RStr='T'; % 用于保存识别出的数字
[temp XT]=size(StrokeTop);
[temp XL]=size(StrokeLeft);
[temp XR]=size(StrokeRight);
%[temp XB]=size(StrokeBottom);
for Ti=2:XT
    if (StrokeTop(Ti)=='C')
        if ((XL==2)&&(XR==2))
 if ((Comp>=3)||((StrokeBottom(2)~='C')&&(StrokeLeft(2)=='C')&& (StrokeRight
(2)=='C')))
                RStr='8';
            else
                RStr='0';
            end

        else if ((StrokeLeft(XL)=='L')&&(StrokeLeft(XL-1)=='P')&& (StrokeLeft
(2)~='C'))
                RStr='2';
            else if ((StrokeLeft(2)=='C')&&(XL>=3)&&(StrokeLeft(3)=='P'))
                    RStr='9';
                else if (XL>2)
                        for Li=2:XL
                            if (StrokeLeft(Li)=='P')
                                RStr='3';
                            end
                        end
                    else if (XL==2)
                            for Ri=2:XR-1
                                if (StrokeRight(Ri)=='P')
```

```
                              RStr='6';
                          end
                      end
                  end
              end
          end
      end
  end
else if  (StrokeTop(Ti)=='V')  % Top
    if ((XR==2)&&(StrokeRight(2)=='C'))  % 数字 3 右端只有一个结构
        RStr='3';
    else  if  ((XR==2)&&((StrokeLeft(2)=='P')||(StrokeLeft(3)=='P')||
(StrokeLeft(XL)=='V')))
        RStr='7';
        else if (XR>2)
            for Ri=2:XR
                if (StrokeRight(Ri)=='P')
                    RStr='5';
                end
            end
            end
        end
    end
    else if ((StrokeTop(Ti)=='L')) % Top
        if ((StrokeRight(XR)=='V')&&(XR<=2))
            RStr='4';
        else
            for Ri=2:XR
                if ((StrokeRight(Ri)=='L')||(StrokeRight(Ri)=='C'))
                    RStr='6';
                end
            end
        end
    end

    end %对应于'V'
  end % 对应于'C'
end % 对应于第一个 for 循环
Numeral=RStr; % 返回识别结果
```

程序结果：

```
9
```

需要指出，上述的识别算法可以为读者提供参考。但在实际应用中环境变化较大，为了提高识别的可靠性和识别的速度需要根据具体情况运用不同的识别算法。

8.3 汽车牌照的自动识别

随着高速公路逐渐普及，我国的公路交通事业发展迅速。而相应的人工管理方式已不能满足实际的需要，微电子、通信和计算机技术在交通领域的应用极大地提高了交通管理效率。目前典型的应用如高速公路自动收费系统、违章车辆自动记录系统和道路车辆数字图像采集系统等。

交通管理的一个常见应用是停车场的自动计费和安全防盗系统。目前，停车场中已经采用了磁卡、IC 卡方式的自动计费设备，但在安全防盗方面考虑不多。这里介绍一种有特色的利用车牌照自动识别技术的停车场安全防盗系统。利用牌照自动识别技术自动监视进出的车辆。当汽车驶进停车场时，识别并记录其牌照号码和其收费 IC 卡号码，而在出口处再次识别其牌照和 IC 卡号并与入口信息进行对比。这样，一般情况下杜绝了犯罪分子利用合法 IC 卡盗窃他人车辆的企图。该系统还可以用在大型车库或重要机关驻地，门卫随时注意进出车辆，以防止不明身份的车辆闯入。使用汽车牌照自动识别系统监视进出车辆的牌照，一旦发现疑问立即提醒保卫人员处理。

8.3.1 车辆管理系统组成

整个系统由车辆传感子系统、IC 卡自动计费系统和汽车牌照识别系统 3 部分组成。

1. 车辆传感系统

车辆传感系统的功能是探测车辆的接近、通过和停留等；采用微处理器系统还可以实现累积记数；探测行驶车速、车道位置等。

许多国家很早就开始研究交通监控系统中这一重要的前端设备。比较成熟的车辆探测器，主要有以下几种：光探测器、微波雷达通过型探测器、测速雷达探测器、压力探测器、声探测器、红外探测器、电磁感应探测器和压敏探测器等。

我国停车场和公路路口建设中应用较多的是红外探测器、电磁感应环探测器。这里采用红外 LED 编码信号的车辆传感器。设置在停车场入口和出口的两对红外发射和接收设备进行车辆检测。利用编码调制信号，增强抗干扰能力。而且一般停车场入口情况比较简单(行人很少)，所以误触发的情况很少。实际应用也证明了红外传感系统具有较强的可靠性。

2. IC 卡自动计费系统

当前在高速公路入口、收费站和停车场已开始普及采用 IC 卡或磁卡或 RF 射频卡的无接触自动计费系统。

以 IC 卡为例，它主要由无源 IC 卡、电磁感应无接触读卡设备和管理软件组成。根据卡的种类不同（如永久卡、临时卡、月卡和小时卡等），IC 卡中存有牌照号码、金额和车主等重要信息。当汽车经过出入口时，司机只要持卡在读卡机附近晃动，IC 卡内部的天线感受到读卡机发出的磁力线，从而为自身提供电源并将自身信息以电磁波方式发回读卡机，完成信息交换。管理软件从而实现自动计费和开闸放行或报警等操作。一般车辆进出时间约在 10~20s 左右，基本可以实现无人值守。

车辆自动识别（Automatic Vehicle Identification，AVI）技术是指识别车辆所具有的车牌、条形码或射频识别标志等特征来自动识别车辆的技术，在现代交通监控及管理中发挥着重要的

作用。汽车牌照识别（License Plate Recognition，LPR）技术是车辆自动识别的重要组成部分，其任务是处理、分析 CCD 拍摄的汽车图像，以自动识别汽车牌照号码。在不影响汽车状态的情况下，大部分 LPR 系统的工作由计算机自动完成，从而降低工作复杂程度。同时，LPR 系统相对于其他两种 AVI 技术——条形码识别（Bar Code Based Identification）和射频识别（Radio Frequency Identification），具有两大优点：首先，LPR 系统不需要在汽车上安装专门的条形码或射频识别标志；其次，LPR 系统是基于视频技术的识别系统，可进行图像回放、检索。因此，LPR 系统具有更为广阔的应用前景。

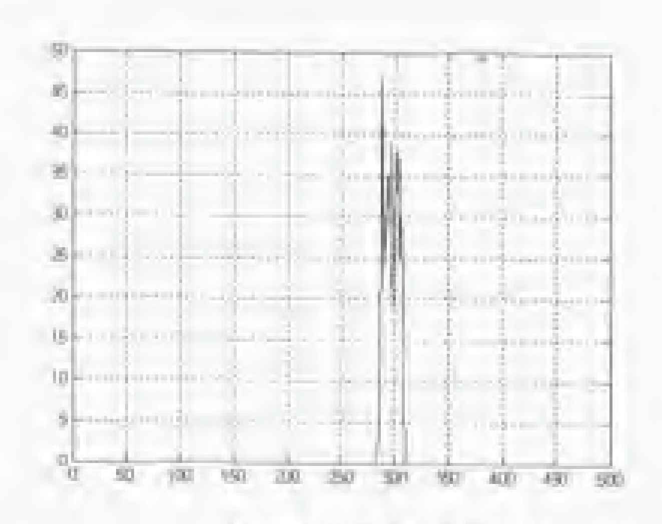
8.3.2　汽车牌照自动识别

汽车牌照自动识别与索书号识别的处理技术相似，关键技术包括车牌定位、灰度（或彩色）图像二值化、字符切分及字符识别等。差别在于两者的应用环境不同，图像特点差异较大，具体所采用的方法不同。

首先要求正确可靠地分割出车牌区域，为此提出了很多方法，如 Hough 变换以检测直线来提取车牌边界区域、使用灰度分割及区域生长进行区域分割，或使用纹理特征分析技术等。Hough 变换方法对车牌区域变形或图像被污染时，失效的可能会急剧增加，而灰度分割则比直线检测的方法要稳定，但当图像中有许多与车牌的灰度非常相似的区域时，该方法也就无能为力了。纹理分割在遇到与车牌纹理特征的其他干扰时，车牌定位正确率也会受到影响。因此，单用一种方法难以达到实际应用的要求，本节利用车牌的彩色信息的彩色分割方法，提高车牌区域定位与分割的正确率。

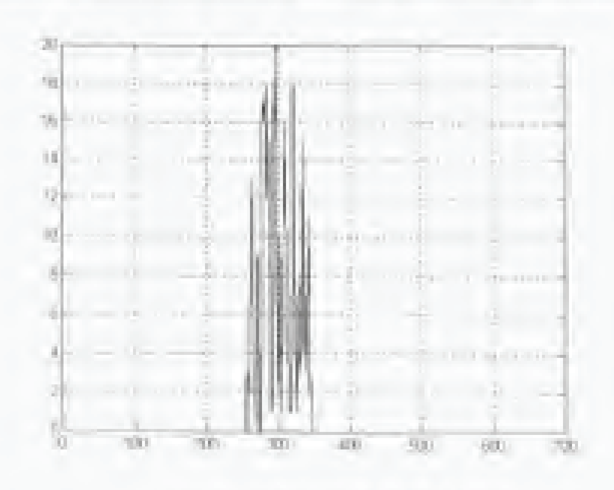


(a) 汽车图像



(b) 蓝色像素点行方向统计

(c) 行方向的车牌区域



(d) 列方向的蓝色像素点统计

(e) 分割出的完整车牌区域

图 8-18

根据车牌底色等有关的先验知识，采用彩色像素点统计的方法分割出合理的车牌区域。下面以蓝底白字底车牌区域为例说明彩色像素点统计底分割方法。CCD 摄像头拍摄的图像一般为 RGB 彩色图像（如图 8-18(a)），确定车牌底色（蓝色）RGB 对应的各自灰度范围，然后行方向统计在此颜色范围内的像素点数量如图 8-18(b)，设定合理的阈值，确定车牌在行方向的合理区域如图 8-18(c)。然后，在分割出的行区域内，统计列方向蓝色像素点的数量如图 8-18(d)，最终确定法确定的完整车牌区域如图 8-18(e)。

【例 8-3】彩色车牌分割

MATLAB 程序代码：

```
I=imread('Car.jpg');
[y,x,z]=size(I);
myI=double(I);
%%%%%%%%%%%% RGB to HSI  %%%%%%%%
tic   % 测定算法执行的时间，开始计时

%%%%%%%%%%%% 统计分析 %%%%%%%%%%%%%%%
%%%%%%%% y 方向 %%%%%%%%%%
Blue_y=zeros(y,1);
for i=1:y
   for j=1:x
if(((myI(i,j,1)<=30)&&((myI(i,j,2)<=62)&&(myI(i,j,2)>=51))&&((myI(i,j,3)<=14
2)&&(myI(i,j,3)>=119)))
% 蓝色 RGB 的灰度范围
            Blue_y(i,1)= Blue_y(i,1)+1;     % 蓝色像素点统计
        end
   end
end
[temp MaxY]=max(Blue_y);         % Y方向车牌区域确定
PY1=MaxY;
while ((Blue_y(PY1,1)>=5)&&(PY1>1))
```

```
    PY1=PY1-1;
end
PY2=MaxY;
while ((Blue_y(PY2,1)>=5)&&(PY2<y))
    PY2=PY2+1;
end
IY=I(PY1:PY2,:,:);
%%%%%%%% X 方向 %%%%%%%%%%
Blue_x=zeros(1,x);                    % 进一步确定 x 方向的车牌区域
for j=1:x
   for i=PY1:PY2
if((myI(i,j,1)<=30)&&((myI(i,j,2)<=62)&&(myI(i,j,2)>=51))&&((myI(i,j,3)<=14
2)&&(myI(i,j,3)>=119)))
              Blue_x(1,j)= Blue_x(1,j)+1;
          end
   end
end
PX1=1;
while ((Blue_x(1,PX1)<3)&&(PX1<x))
    PX1=PX1+1;
end
PX2=x;
while ((Blue_x(1,PX2)<3)&&(PX2>PX1))
    PX2=PX2-1;
end
PX1=PX1-2; % 对车牌区域的修正
PX2=PX2+2;
Plate=I(PY1:PY2,PX1-2:PX2,:);
t=toc % 读取计时
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure,imshow(I);
figure,plot(Blue_y);grid
figure,plot(Blue_x);grid
figure,imshow(IY);
figure,imshow(Plate);
```

程序结果如图 8-18(e)所示。

由于光照条件的随机变化和噪声干扰等原因，一般在自然场景中实现理想分割比较困难，为了提高彩色分割算法的可靠性，提出了用 HSI 彩色空间和神经网络进行彩色分割，对蓝色像素点进行统计。

为提高系统识别率，采用由多级分类器集成的识别方案。由数字“1”特征检测、汉字识别神经网络、英文数字识别神经网络及一个混合分类神经网络组成字符识别神经网络，分别专门用于捡出数字“1”、识别汉字、英文、数字字符。对于 D 和 0、6 和 8、2 和 Z、A 和 4 等易混淆的字符用一个混合分类神经网络来专门分别。对于数字“1”检出是基于以下合理假设，有较大的宽度比，笔划垂直，没有环圈，字符中心线方差小。字符识别神经网络的输入可以取

原彩色图像中字符区域的灰度图像或彩色分割后的字符区域灰度图像，字符图像的尺寸识别前进行归一化处理。为提高系统的识别率及置信度，需对分类器的输出结果按车牌编号规则及车牌数据库进行校验。

8.4 商标的自动翻译

随着我国2008年奥运会和2010年世界博览会的申办成功，在今后的几年里，将会有越来越多的外国旅客来我国旅游。而对于外国游客来说，汉语与它们母语的不同，将成为他们旅游的最大问题。

虽然，目前的便携式电子词典如文曲星等发展成熟，但是外国游客却很难将他们感兴趣商品名称上标注的文字，地名等汉语文字直接输入电子词典。本节提出利用便携式CCD摄像头拍摄文字图像，利用OCR技术自动识别后输入电子词典，然后，将汉语文字翻译成英语或其他语言，实现商标的自动翻译，为外国游客的旅游观光提供方便。

8.4.1 商标自动翻译系统的组成

商标自动翻译系统的组成如图8-19所示。具体的工作流程如下：便携式CCD摄像头拍摄商品附近标注的商品名称文字，经文字图像分割后，利用OCR文字自动识别技术识别文字，识别出的文字送入电子词典进行翻译，最后将翻译结果以英语或其他语种形式显示给游客。

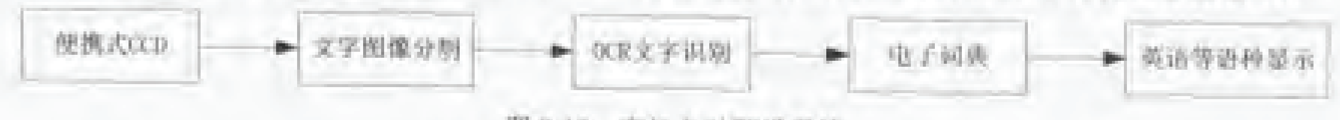


图8-19 商标自动翻译系统

到目前为止，商标自动翻译系统中所涉及到的便携式CCD、电子词典和OCR文字识别技术都已经非常成熟，而如何从商品的自然场景图像中准确分割出文字图像，依然是一个关键的技术和难点。

8.4.2 商标文字图像的分割

商标文字的颜色多种多样，有黑色、白色、红色等等，商品更是五颜六色，所以无法用颜色特征分割。但与索书号、汽车牌照等文字图像相比，游客可以现场调整商标文字图像的区域范围，将其限制在一个较小的范围内，使得图像的内容比较简单，即主要是商标图像，如图8-20所示。



图8-20 商标图像

本节根据文字图像区域内文字和背景的对比度强烈，而且文字笔划具有正负梯度交替变化的特征，利用最大梯度差检测可能的行，检测文字图像行经合并后形成文字图像区域。

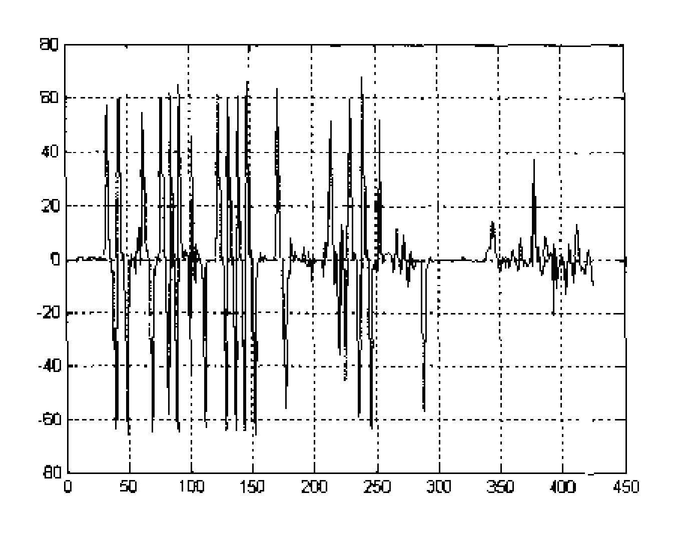


图 8-21　文字图像的梯度曲线

1. 图像的梯度计算

对CCD拍摄的灰度图像，计算其相应的梯度矩阵。由于文字图像与背景的对比度强烈，其梯度值较大。图8-20商标图像中的某一行文字图像像素的梯度值变化曲线图，如图8-21所示，文字图像区域的梯度绝对值明显大于非文字的图像区域。

2. 梯度差

从背景进入文字笔划处的梯度值 G_{In} 与从笔划进入背景的梯度值 G_{Out} 符号相反，梯度的绝对值接近相等。

在文字笔划的附近选取某一区域，求得区域内的最大梯度值 G_{max} 和最小梯度值 G_{min}，则该区域内的最大梯度差MGD（Maximum Gradient Difference），即为

$$MGD = G_{max} - G_{min} \tag{8-30}$$

设区域大小为 $n\times1$。图8-21所示的梯度曲线相应的最大梯度差曲线如图8-22所示。

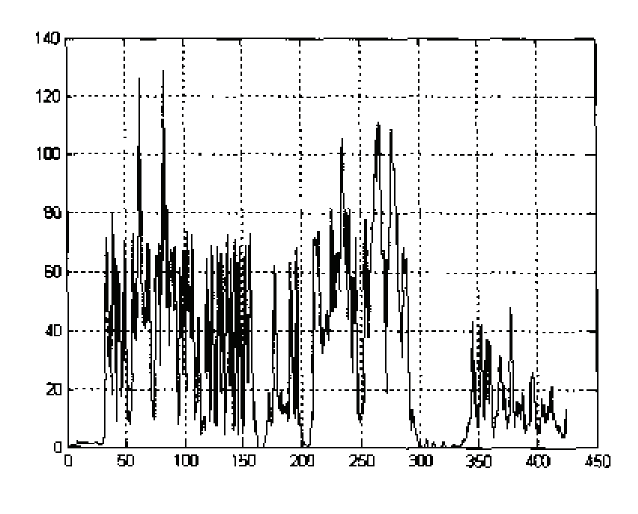


图 8-22　最大梯度差曲线

3. 可能文字图像行检测

设定阈值，MGD 大于阈值的为可能文字图像行。由于噪声干扰和部分商品图像的存在，非文字图像区域中有时也存在个别 MGD 大于阈值的情况。因此，选用 MGD 大于阈值的次数统计确定文字图像。

最终分割出的商标文字图像如图 8-23 所示。

心脑血管药类

图 8-23　商标文字图像分割结果

注意：分割出的文字图像右侧之所以有较大纯背景区域，是由于商标的标签纸有水平方向的残缺，使得均匀背景纹理特性被破坏，从而具有与文字图像相似的最大梯度差 MGD 的特点，所以也被视为文字图像。

当然，由于分割出的文字图像仅含有文字，因此结果也算理想，但如果含有其他图像，则分割将失败。

【例 8-4】MGD 商标文字分割

MATLAB 程序代码：

```
I=imread('brand02.jpg');
HS_I=double(I);
tic
G=gradient(HS_I);  % 计算梯度值
[y,x]=size(I);
T=60;
%%%%%%% 商标文字分割 %%%%%%%%%%
n=30;
GY=zeros(y,1);
for j=1:y
   for i=1:x-n
        Max=max(G(j,i:i+n));
        Min=min(G(j,i:i+n));
        MGD(j,i)=Max-Min;         % 计算最大梯度差
        if (MGD(j,i)>T)
           GY(j,1)=GY(j,1)+1;  % MGD 大于阈值的次数统计
        end
   end
end
[temp MaxY]=max(GY);
PY1=MaxY;
while ((GY(PY1,1)>=n)&&(PY1>1))  % 次数阈值检测
      PY1=PY1-1;
end
PY2=MaxY;
```

```
while ((GY(PY2,1)>=n)&&(PY2<y))
      PY2=PY2+1;
end
%%%%%%%%%% X 方向的分割 %%%%%%%%%%
n=15;
GX=zeros(1,x);
for i=1:x
   for j=1:y-n
        Max=max(G(j:j+n,i));
        Min=min(G(j:j+n,i));
        MGD(j,i)=Max-Min;
        if (MGD(j,i)>T)
           GX(1,i)=GX(1,i)+1;
        end
     end
 end
 PX1=1;
while (((GX(1,PX1)<=n)||(GX(1,PX1+1)<=n))&&(PX1<x)) % 列方向保证同时相邻的两列满
足次数要求
      PX1=PX1+1;
end
PX2=x;
while (((GX(1,PX2)<=n)||(GX(1,PX2-1)<=n))&&(PX2>PX1))
      PX2=PX2-1;
end
HS_I=uint8(HS_I);
IY=HS_I(PY1:PY2,PX1:PX2);
t=toc
%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure,imshow(HS_I);
figure,plot(MGD(40,:));grid
figure,plot(MGD(140,:));grid
figure,imshow(IY);
```

程序结果如图 8-22 所示。

第 9 章 AGV 视觉导引车路径识别

随着生产技术的发展和自动化程度的提高，传统制造业的生产方式发生了深刻的变化。为节约成本、缩短生产周期，柔性生产系统和工厂自动化等先进的生产方式逐渐发展起来。其中，AGV 则成为物流系统环节搬运设备的代表。

AGV 的导引方式决定着由其组成的物流系统的柔性，也影响着系统运行的可靠性和组态费用。本章将结合实例，详细介绍图像处理与识别技术在 AGV 自动导引车中的应用。

9.1 AGV 及其视觉导引技术简介

9.1.1 AGV 概述

AGV（Automated Guided Vehicle），即自动导引车，指装备有电磁或光学等自动导向设备，能够沿规定的导向路径行驶，在某一位置自动进行货物的装载，自动走到另一位置，且自动完成货物的卸载，并具有安全保护以及各种移栽功能的全自动运输装置，属于智能车辆的范畴。

AGV 自动导引车一般由控制系统、机械机构和电源装置 3 部分组成。机械机构包括车体总成、驱动/转向总成、移栽总成和安全机构 4 个部分。电源装置包括：蓄电池、逆变器和充电装置 3 个部分。控制系统包括：计算机、传感器、控制器、通信装置和检测报警装置等几部分组成。通过传感器检测当前小车行走方向的信息，并由计算机控制调整，使小车按照预期的方向行驶。

9.1.2 AGV 的发展及其应用

AGV 最早在上世纪 50 年代初产生于美国。早期研制 AGV 的目的是为了提高仓库运输的自动化水平，应用领域也仅仅局限于仓库内的物品运输。其控制器采用真空电子管，体积较大，且功能受到限制，实质上只是简单动力化的拖车或载货车，性能较差。第一代 AGV 是只能沿着预先设置的路线运行。由于当时技术和经济条件的限制，AGV 的发展速度极为缓慢。

20 世纪六七十年代早期，集成电路电子技术在 AGV 上的应用促进了 AGV 的自动化及控制性能的提高，使控制系统功能复杂的 AGV 的开发有了现实可行性。在这一时期，欧洲地区的公司对托盘的尺寸与结构进行了标准化，这一措施有力地促进了 AGV 技术在欧洲的快速发展。

20 世纪七八十年代，随着集成电路电子技术和计算机技术的发展，微处理器开始普遍应用于 AGV 领域，AGV 的自动化和智能化程度大大提高。为了适应快捷的现代生产需要，许多企业开始采取基于 AGVS 为载运工具的柔性制造系统 FMS（Flexible Manufacture System），柔性装配系统 FAS（Flexible Assemble System）。1974 年，瑞典的 Volvo Kalmar 轿车装配工厂

与 Schündler-Digitron 公司合作，开发研制出各种可装载轿车车体的 AGVS，并将多台改造的 AGV 组成了汽车装配线。Kalmar 工厂采用 AGVS 获得了明显的经济效益，使得许多西欧国家纷纷仿效，AGVS 的研究由此在西欧许多国家得到了迅速的发展。1984 年，General Motors 公司建立了其第一个基于 AGVS 的轿车柔性装配线，到 1986 年，使用的 AGV 达到 1407 台，General Motors 公司成为 AGV 的最大用户，带动了 AGV 在工业领域的大规模应用。

进入 20 世纪 90 年代以来，随着电子、计算机、通信、人工智能、图像处理技术的飞速发展，AGV 的自动化和智能化程度进一步提高，各种满足不同需要的 AGV 也应运而生。到目前为止，AGV 已被广泛应用于机械、物料装配、电子、化工、烟草、医疗、钢铁、汽车、民航、食品、核材料和感光材料等行业。图 9-1(a)所示是 AGV 应用于汽车工业的实例，图 9-1(b)所示是 AGV 应用于报纸印刷业的实例。



(a) 应用于汽车工业的 AGV



(b) 应用于报纸印刷业的 AGV

图 9-1

9.1.3 AGV 导引技术简介

AGV 的控制技术已日趋成熟，决定 AGV 是否能进一步应用于更加复杂、恶劣工作环境的关键技术是其导引技术。

根据 AGV 导引信息的来源，导引方式可分为外导式和内导式。外导式是指在车辆运行路径上设置导引信息媒体（如带有变频感应电磁场的导线、磁带或色带等），由安装在车上的传感器检测导引信息的特性（如频率、磁场强度、光强度等），再将此信息经过处理，控制车辆沿导引路线行驶。内导式是指在车辆上预先设定运行路径坐标，在车辆运行中实时检测车辆当前位置坐标并与预先设定值相比较，控制车辆的运行方向，即采用所谓的坐标定位原理，因此内导式方法又可称为参考位置设定法。另外，根据 AGV 导引线路的形式，导引方式又可分为有线式和无线式两种。如表 9-1 所示列出了 AGV 的导引方式。

表 9-1　　AGV 的导引方式计方

分类	按导引信息的来源		按导引线路的形式	
导引方式	外导式	内导式	有线式	无线式
导引方法	电磁导引	坐标导引	电磁线路	超声波导引
	超声波导引	惯性导引	磁带导引	激光导引
	激光/红外线导引	自主导航	色带线路	坐标识别
	光学导引		网格线路	惯性导引
	标线导引		标线线路	自主导航

电磁导引是目前的无轨 AGV 主要采用的方法。它需在 AGV 要行走的路线下面埋设专门电缆线，其中通以一定频率的交变电流形成交变的电磁场，同时在 AGV 车体上对称设置一对电磁传感器（感应线圈），利用电磁感应原理，通过检测电磁感应信号的强度，引导 AGV 沿着埋设的路线行驶。该方法可靠性高，成本低，但对地面的平整度要求高，运行路径改变困难。

光学导引的基本原理和方法同电磁导引相似。一般是在运行路径上铺设一条具有稳定反光率的色带。车上设有光源发射和接收反射光的光电传感器，通过对检测到的信号进行比较，调整车辆的运行方向。

激光红外线导引是在 AGV 上装备可发射和接收激光/红外线的扫描器，导引区域的四周按要求布置足够的反射板，AGV 的控制系统中存储有各块反射板的数据及运行路径的信息。AGV 运行时实时接收固定设置的 3 点定位激光/红外线信号，通过计算测定其瞬时位置和运行方向，然后与控制系统中存储的运行路径信息进行比较，引导 AGV 运行。激光/红外线导引技术的导向与定位精度较高，且提供了任意路径规划的可能性。但成本高，传感器和发射或反射装置的安装复杂，位置计算也复杂。

惯性导引采用陀螺仪检测 AGV 的方位角，并根据从某一参考点出发所测定的行驶距离来确定当前位置，通过与已知的地图路线进行比较来控制 AGV 的运动方向和距离，从而实现自动导引。其主要优点是技术先进、准确性高和灵活性强，便于组合和兼容，适用领域广。缺点是成本较高，维护保养等后续问题较难解决，地面也需磁性块作辅助定位。

9.1.4 视觉导引技术

视觉导引又称图像识别导引，它分为无线式和有线式两种。无线式视觉导引利用 CCD 系统动态摄取运行路径周围环境图像信息，并与拟定的运行路径周围环境图像数据库中的信息进行比较，从而确定当前位置及对继续运行路线做出决策。有线式视觉导引根据路面或路边的明显路径标识线，通过车载 CCD 摄像机动态摄取路面图像，经车载计算机处理识别出路径标识线，并判断车辆纵向轴线偏离标识线的距离及其与标识线间的夹角，通过控制转向系统使车辆的实际行驶路线与路径标识线的偏差保持在允许的范围内。

有线式视觉导引技术既具有获取的信息容量大、路径的设置和变更简单方便、系统柔性好等优点，又具有现实应用的可能和广阔的应用前景，是当前智能车辆导引技术研究的主流方向和发展趋势。

9.2　路径摄像系统

AGV的视觉导引系统通过CCD摄像系统拍摄路径标线的图像，摄像系统的硬件性能将决定路径识别的精度和实时性，以及后续图像处理和识别算法性能。

9.2.1　AGV视觉导引的硬件体系结构

AGV视觉导引系统的硬件结构如图9-2所示。由图可知，视觉导引系统按照功能可分为3个层次：感知层实时地采集与导引控制有关的信息，包括路面图像信息和转向轮当前的转动位置信息等，主要由CCD摄像偷、图像采集卡、光电编码器等组成；决策层处理感知层采集到的信息，并根据处理结果依据某种控制策略发出控制指令，主要由一台工业控制计算机、一块集模拟量、数字量的输入输出等功能于一体的多功能板及其接线端子板构成；执行层主要由电机及其控制器、减速机构等构成，接受决策层发出的指令并执行相应的操作。

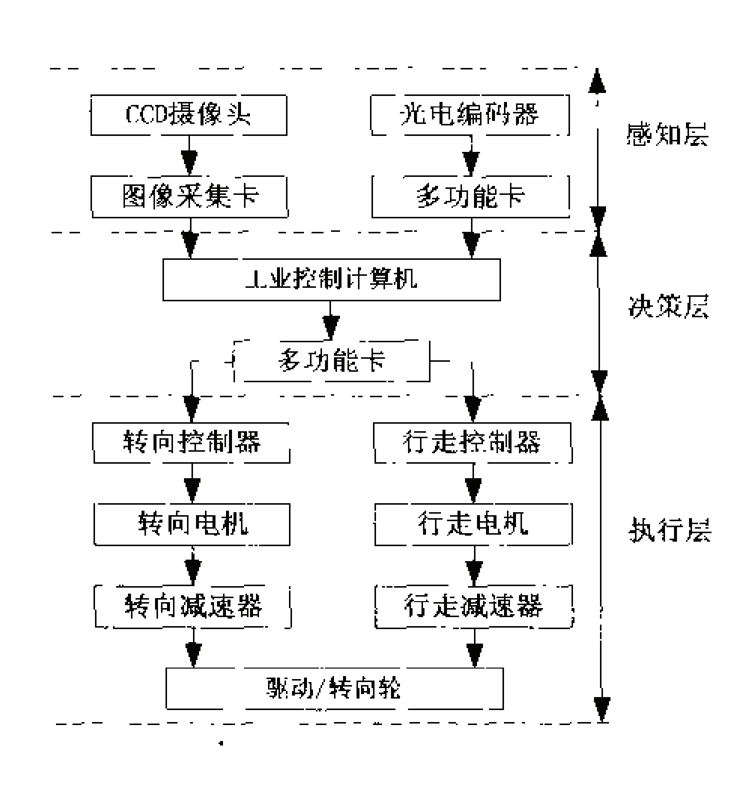
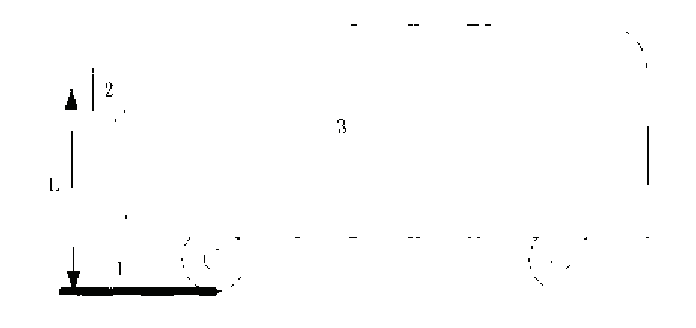


图9-2　AGV视觉导引系统的硬件结构

9.2.2　CCD摄像系统设计

1. CCD安装与选择

CCD摄像头的安装位置影响输入的路径标线的图像特征，安装位置应保证得到画面稳定、满足系统控制要求的路径标线图像。一般将摄像机安装在车身上，摄像机振动幅度小，工作环境好，输入图像稳定、清晰，输入图像场景大，分析得到的控制信息超前于驱动轮控制，对系统反应速度的要求降低。本系统将CCD安装在车体前部的车体纵轴线上，摄像机主光轴与地面垂直，如图9-3所示。



1. 路径标线　2. 摄像头　3. AGV 车体

图 9-3　AGV 路径摄像示意图

路径标线经光源照明后，经成像物镜，将其像成在 CCD 光敏阵列上面，如图 9-4 所示。

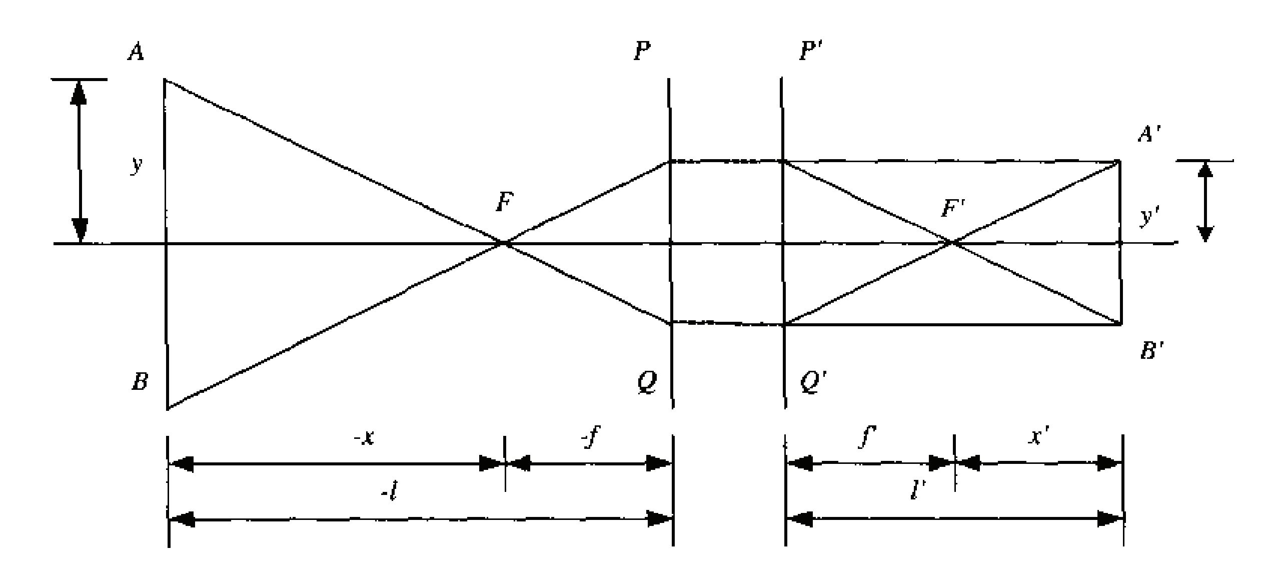


图 9-4　CCD　光学系统成像示意图

物空间的位置坐标 x 以物方焦点 F 为坐标原点；以光轴为横轴 x，并以入射线的传播方向为正方向。纵坐标用物高 y 来表示。像空间的位置坐标 x'，以像方焦点 F' 为坐标原点；以光轴为横轴，并以出射线的传播方向为正。纵坐标用像高 y' 来表示。物高和像高以光轴为起始轴，从下向上为正。为了成像关系几何表示方便，光路图中表绝对量。

摄像系统的设计，需要确定器光学成像系统的特性参数。成像光学系统的设计比较复杂，精密光学仪器的加工更为麻烦。一般来说，通常选用成熟的标准化产品。CCD 摄像头的选择主要考虑以下几个因素：

（1）放大倍数 β

$$\beta = \frac{2y'}{2y} \tag{9-1}$$

式中，$2y' = N\delta'$，为 CCD 敏感面尺寸。N 表示 CCD 的像素点单元数，δ' 表示像素点单元的宽度。

（2）物镜成像的方程组

$$\begin{cases}\beta=\dfrac{l'}{l}\\ l'-l=L\\ \dfrac{1}{l'}-\dfrac{1}{l}=\dfrac{1}{f'}\end{cases} \tag{9-2}$$

式中，L 由结构尺寸决定如图 9-3 和 9-4 所示；f'为物镜的焦距；l'和 l 为成像系统的外形尺寸。

（3）视角 $2w$

$$tgw=\frac{y'}{x'} \tag{9-3}$$

式中，$x'=l'-f'$为 CCD 光敏面到物镜像方焦点 F'的距离。

一般来说，测量范围和测量精度是选择 CCD 器件的主要依据。设待测路径的宽度为 W，允许测量误差为 Δ。CCD 器件的主要参数为像素点数，线阵 CCD 为 N 像素，面阵 CCD 为 $M\times N$（设测量方向的像素点数为 N），它主要决定测量范围；相邻像元的中心距 C，它主要决定测量误差。则必须满足下述两式：

$$\beta\times W<N\times C \tag{9-4}$$

即路径成像在 CCD 敏感尺寸范围内。

$$C<\beta\times\Delta \tag{9-5}$$

即一个 CCD 像元所代表的测量值应该小于测量误差，否则无法保证测量精度。另外，CCD 元器件主要工作参数是工作频率。

在实际应用中，摄像系统的设计并不能用上述公式完成设计。主要原因是：①系统的设计还受后续算法的影响；例如，分辨率低的 CCD，虽然测量误差较大，但图像的数据量小，图像处理所需要的时间就少，可以采用高级算法来弥补其测量误差；②相同测量要求可以采用多种不同的方法实现；如某些场合可以选用线阵 CCD，也可以选择面阵 CCD；③CCD 器件的像素点数是分级的，一般为 2^m，m 为正整数，而且常用的产品也就是固定的 4~5 个系列。

（4）目前，CCD 技术和计算机技术发展迅速，一般的工业要求都能得到满足，如高速高分辨率的 CCD 已经有成熟的产品，但价格昂贵。设计的关键是系统要有较高的性能价格比。

经常用的设计方法，实际上是根据设计公式进行估计，初步确定放大倍数β，安装结构尺寸 L 以及 CCD 的像素点和像元宽度的选择范围，然后选择一款合适的 CCD，进行调试。通过调整焦距使图像保证清晰，并保证路径标线在 CCD 摄像范围内，拍摄一幅图像，分析其测量精度，使其满足测量要求。否则，可以适当地调整结构尺寸 L 之后，再调整摄像头焦距。虽然高分辨率地 CCD 测量精度会比较高，但是图像数据量的将急剧增加，对系统其他硬件地要求也相应提高，否则算法处理时间过长，无法满足实时性能的要求。因此，建议在满足测量要求的前提下，尽量考虑采用分辨率较低的 CCD。本系统中选用分辨率为 640×480 像素点，像素点单元尺寸为 11μm 的 CCD。

2. 图像采集卡

图像采集卡的任务是将 CCD 摄像机输出的视频信号（模拟信号）转化为方便计算机使用

的数字信号。图像采集卡的工作过程可以描述为：实时采集 CCD 输出的视频信号，将此信号经 A/D 转换后以数字图像的形式存放在图像单元的一个或两个通道中，通过计算机发出指令，将某一帧图像暂存在图像存储通道中，即采集或捕获了一帧图像，计算机即可对采集的图像进行各种处理。采集卡上的 D/A 转换电路自动将图像实时显示在图像监视器上。

图像采集卡的性能应该与 CCD 的相匹配，主要设计的参数：

（1）图像传输速度：设 CCD 的摄像频率为 F 帧/秒，每幅图像为 $M\times N$ 像素点，则图像数据传输速度 v 必须满足

$$v > F\times M\times N \tag{9-6}$$

如果图像为 RGB 彩色图像，则图像传输速度的要求相应提高为原来的 3 倍。

（2）A/D 转换器的位数

如无特殊要求一般均采用 8 位 A/D 转换器。由于图像的数据量大，一般均采用告诉 A/D 转换器。A/D 转换频率 f 与图像数据传输速度相似，必须满足

$$f > F\times M\times N \tag{9-7}$$

如果图像为 RGB 彩色图像，A/D 转换频率同样需要提高为原来的 3 倍。

以上两个参数为硬指标，通常取值还需要一定的预留量。这也说明了，像素点增多对图像采集卡性能要求的影响。图像采集卡根据实际应用场合的具体需要还有其他的一些技术要求，例如缓存的大小，总线接口类型以及支持的制式等。

本系统采用美国 NI 公司的 NI PCI-1411 彩色图像采集卡，图像最大分辨率为 879×978，支持 PAL、NTSC、RS-170 等多种制式。它在板上配置了专门的处理器，可直接在板上进行从 RGB 到 HSL 空间的颜色转换，直接输出 HSL 格式的图像；可在板上编程指定感兴趣的采集区域。板上内存 16MB。采用 PCI 总线，能实时传送数字视频信号到显示存储器，或系统存储器，或直接存于硬盘，数据的传输过程由图像采集卡控制，无需 CPU 参与，图像的传输速度最大可达 132MB/s。

9.3 路径图像识别

路径图像的识别是指将路径标线从图像背景中分离出来，以便进行小车行驶偏差的测量。一般路径图像内容相对比较简单，如图 9-5 和图 9-6 所示。通常是以路径为主，背景是简单的路面，另外存在一些随机杂物的干扰，所以路径识别相对简单。主要问题是现场光照环境不稳定，路径褪色以及路面光照反射能力的差别（如路面不平整，路面上的污迹）造成路径图像灰度不均匀。本系统的路径识别的目的是为了检测小车行驶方向的偏差。因此，路径识别的任务主要是将摄像头拍摄的灰度或彩色图像转换成二值图像，其中路径标线以白色表示，背景及其他图像内容用黑色表示，以便下一步进行小车行驶的测量。其实质与图像分割中二值化相似。

9.3.1 路径图像的特征

如图 9-5 和图 9-6 所示，分别为路径图像的灰度图像和彩色图像。在 AGV 运行过程中，实际的导引路径标线基本上是由直线再加少量一定曲率半径的弧线（转弯处）组成的。由于 CCD 摄像机安装位置距离地面相对较近，拍摄到的图像是近视野图像，因此，可以将图像中

的路径都近似看作直线来处理，这样处理既能保证一定的准确性又大大减少了计算量。



图 9-5　路径的灰度图像



图 9-6　路径的彩色图像

因此，首先路径图像的形状特征为一连续的、具有一定宽度的直线；其次，路径与路面背景具有强烈的对比度。在灰度图像中表现为灰度值差别较大，在彩色图像中表现为颜色差异鲜明。最后，还有一部分辅助特征，即一幅图像中有且仅有一条路径，并在路径伸长方向，贯穿整幅图像等。

灰度图像和彩色图像各有其特点，在不同环境都有一定实用性，本节分别采用不同的方法对灰度和彩色的路径图像进行识别处理。

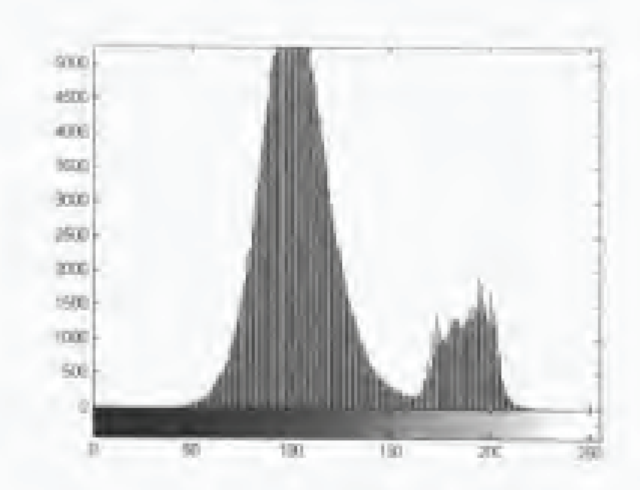
9.3.2　灰度图像的路径识别

1. 最优阈值分割

最优阈值分割算法已经在第 7 章中详细介绍，并且在细胞的阈值分割中应用，此处不再详细介绍。在进行阈值分割前，需要对其灰度图像的直方图统计进行分析，以便确认能否进行阈值分割。路径灰度图像及相应直方图统计如图 9-7(a)和图 9-7(b)所示。



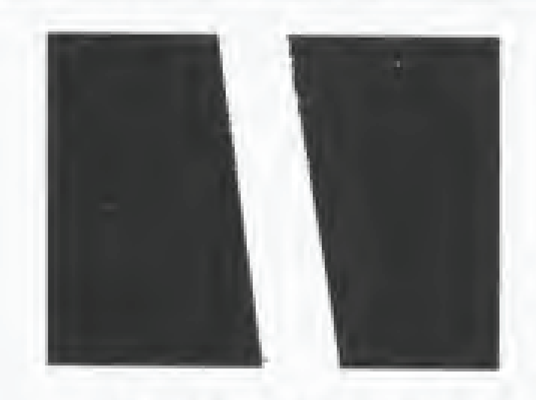
(a) 路径灰度图像



(b) 路径灰度图的直方图统计



(c) 最有阈值分割结果



(d) 形态学处理的路径二值化图像

图 9-7

此处，统计直方图上明显存在两个峰值，峰值和峰谷明显，而且只有两个峰值一个峰谷，恰好对应于路面背景和路径目标，可以用直接阈值进行分割。但实质上，AGV 小车在运行的过程中，不同位置的光照条件和路面对光的反射能力有着差别，因此灰度直方图的峰谷对应的灰度值是动态变化的。固定阈值对某几幅图像得到满意结果，而在整个运行过程中，会产生较大误差。采用最有阈值分割的根据能够灰度统计峰谷处灰度值的变化自适应地求得最佳分割阈值，所以可以采用最优阈值分割进行路径阈值分割。最优阈值为 142，分割结果如图 9-7(c)所示。实际上，这个阈值也并非最优，称之为满意阈值较为合适。

2. 形状分析

经过阈值分割后，还存在许多离散的小斑点，主要是由于路面差异引起的。而且，路径边缘非常毛糙，影响路径的定位。根据路径的形状特征，采用数学形态学的“开”运算进行处理，结果如图 9-7(d)所示。消除离散斑点的同时，路径边缘相对变得比较光滑。当然，部分不光滑的边缘可以在最后路径测量、定位阶段，采用最小二乘法拟合直线的处理方法消除。

阈值分割的算法比较简单，运算速度较快，具有较高的实时性。但是，它对图像的质量要求较高，路径图像具有较高灰度均匀一致性，灰度统计直方图具有明显的双峰。因此，在环境较理想的地方，采用阈值分割的简单方法识别路径，可以降低整个系统硬件性能的要求，或者可以提高系统的实时性。

【例 9-1】灰度 AGV 路径识别（MATLAB 程序代码）

```
J= imread('RoadS1.jpg');
[x,y]=size(J);
I=double(J);
z0=max(max(I));                 % 求出图像中最大的灰度
z1=min(min(I));                 % 最小的灰度
T=(z0+z1)/2;
TT=0;
S0=0; n0=0;
S1=0; n1=0;
allow=0.5;                      % 新旧阈值的允许接近程度
d=abs(T-TT);
count=0;                        % 记录几次循环
while(d>=allow)                 % 迭代最佳阈值分割算法
```

```
        count=count+1;
        for i=1:x
            for j=1:y
                if (I(i,j)>=T)
                    S0=S0+I(i,j);
                    n0=n0+1;
                end
                if (I(i,j)<T)
                    S1=S1+I(i,j);
                    n1=n1+1;
                end
            end
        end
        T0=S0/n0;
        T1=S1/n1;
        TT=(T0+T1)/2;
        d=abs(T-TT);
        T=TT;
    end
    Seg=zeros(x,y);
    for i=1:x
        for j=1:y
            if(I(i,j)>=T)
                Seg(i,j)=1;                  % 阈值分割的图像
            end
        end
    end
    J0=Seg;
    SE=strel('rectangle', [6 4]);           % 结构定义
    IM=imopen(J0,SE);                 %  "开" 运算
    figure,imshow(J);    % 图 9-7(a)
    figure,imhist(J);      % 图 9-7(b)
    figure,imshow(Seg);  % 图 9-7(c)
    figure,imshow(IM);  % 图 9-7(d)
```

程序结果如图 9-7(a)~(d)所示。

9.3.3 彩色图像的路径识别

在工业环境中，光照条件往往无法控制，或者需要昂贵的成本，而且路面的表面特性受油垢，污迹和不同程度损坏影响差别较大，路径的灰度值往往在局部区域内变动较大，或者灰度统计没有明显的双峰时，用简单的阈值分割方法和形状分析很难准确识别路径。图 9-5 的路径灰度图像的灰度统计直方图如图 9-8 所示。而且还随时可能有其他杂物（如纸屑等）出现在路面上，路径的图像内容也变得有些复杂。

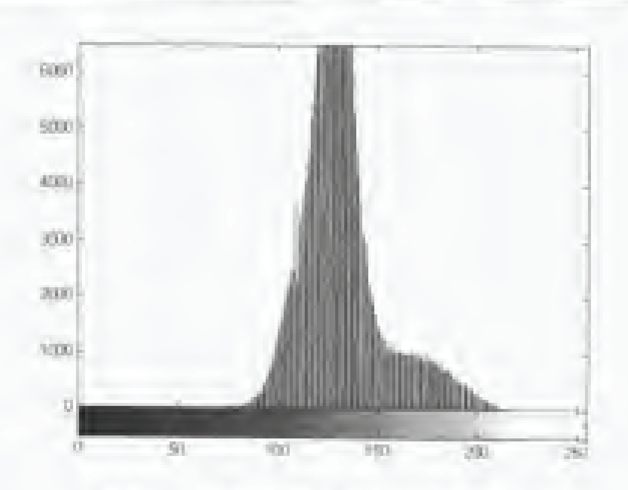
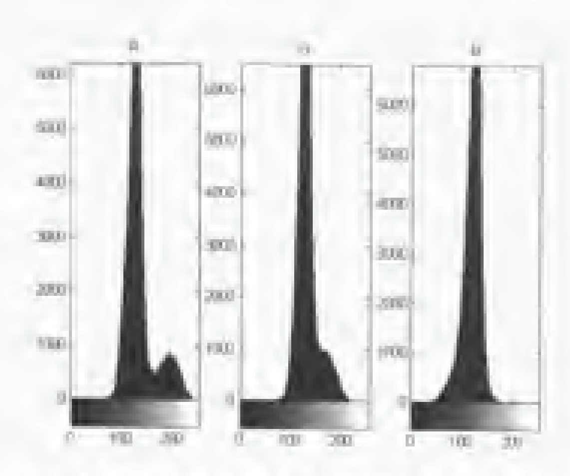


图 9-8　路径灰度图像 9-5 的灰度直方图

利用颜色特征识别路径图像，可以减少光照的不均匀和路面光反射能力差别的影响，还能对随机杂物有一定的抗干扰能力。

1. 颜色聚类的路径识别

图 9-5 的灰度路径图像改用彩色图像表示，如图 9-6 所示。由于图像中内容比较简单，只有两类，路面背景和路径，且各自在颜色空间的距离较大，采用 Euler 距离的颜色聚类，能够将路径准确识别出来。



(a) 彩色路径图像图 9-6 的 RGB 统计直方图



(b) 颜色聚类的路径识别



(c) "闭"运算处理后的完整路径图像

图 9-9

首先，类的数目确定为 2，即路径和路面。然后，确定聚类中心。根据先验知识已知路径的颜色为黄色，在 RGB 空间的位置坐标为(1,1,0)。但是由于路径颜色的褪色，实际图像中黄色坐标空间略有偏移。通过分析 RGB 图像 3 个分量的统计直方图，如图 9-9(a)所示。设定初始聚类中心，由于 AGV 小车在行驶过程中，路径和路面背景颜色会有波动，相应的聚类中心也会发生抖动。但动态聚类的计算量比较大，实时性比较差。通过对行驶全过程的聚类分析，设定一个满意的固定聚类中心：路径为(200,160,0)，路面为(105,110,110)。

利用 Euler 公式，分别计算图像中某点的 RGB 分量距离路径颜色聚类中心和路面颜色聚类中的距离，即：

$$d_1 = \sqrt{(R-R_1)^2 + (B-B_1)^2 + (C-C_1)^2} \tag{9-8}$$

式中，(R_1,G_1,B_1)为路面背景的聚类中心。

$$d_2 = \sqrt{(R-R_2)^2 + (B-B_2)^2 + (C-C_2)^2} \tag{9-9}$$

式中，(R_2,G_2,B_2)为路径的聚类中心。比较 d_1 和 d_2 的大小，若 d_2 小于 d_1，则该像素点被识别为路径。路径图像 9-6 经过颜色聚类处理后得到的图像如图 9-9(b)所示。

受图像路径颜色的褪色不均匀的影响，路径上出现了许多黑斑，影响路径后续处理的检测与定位。同样，利用路径形状特征，进行形态学“闭”运算处理。形态学运算采用的结构模板通过试验设置。在保证消除路径黑斑的前提下，尽量使用较小的结构模板，减少不必要的运算量，本算法采用 20×15 的矩形结构。“开”运算处理的图像如图 9-9(c)所示。

【例 9-2】彩色 AGV 路径识别（MATLAB 程序代码）

```
I=imread('RoadG2.jpg');
I1=I(:,:,1);
I2=I(:,:,2);
I3=I(:,:,3);
[y,x,z]=size(I);
d1=zeros(y,x);
d2=d1;
myI=double(I);
I0=zeros(y,x);
for i=1:x
   for j=1:y
d1(j,i)=sqrt((myI(j,i,1)-105)^2+(myI(j,i,2)-110)^2+(myI(j,i,3)-110)^2);     %
Euler 距离计算
d2(j,i)=sqrt((myI(j,i,1)-200)^2+(myI(j,i,2)-160)^2+(myI(j,i,3)-0)^2);
        if (d1(j,i)>=d2(j,i))
             I0(j,i)=1;          % 颜色聚类
        end
   end
end
J=I0;
tic
SE=strel('rectangle', [20 15]);     % 形态学“闭”运算
IM=imclose(J,SE);
```

```
toc
figure(1);imshow(I);
figure(2);subplot(1,3,1);imhist(I1);title('R');
subplot(1,3,2);imhist(I2);title('G');
subplot(1,3,3);imhist(I3);title('B');
figure(4);imshow(I0);
figure,imshow(IM);
```

程序结果如图 9-9(a)～(c)所示。

2. HIS 彩色空间的彩色分割

在 AGV 组成的自导引车辆系统中，经常用不同的颜色来表示不同的路径，不同的路径对应于 AGV 不同的搬运工作。多种颜色的路径，很难用固定聚类中心的方法识别。一方面，每一个像素点的距离计算量急剧增加。有 n 种不同颜色的路径，就要计算 n+1 个距离（其中包括路面背景距离）。其次，固定聚类中心的准确性降低。

由于 RGB 彩色空间的 3 个分量之间有很高的相关性，而且分量值都不同程度地受光照影响，直接利用识别路径比较困难。将 RGB 彩色空间转换到 HSI 彩色空间，H 表示不同的颜色，如黄、红、绿，S 表示颜色的深浅，如深红、浅红，I 表示颜色的明暗程度，主要受光源强弱影响。

在路径识别中，光照条件变化难以控制，不用 I 分量。在彩色空间转换的计算中，为了减少计算量，不做 I 分量值的计算。利用 H 和 S 分量值进行多阈值的分割，识别出不同颜色的路径。

$$H=\cos^{-1}\left[\frac{(R-G)+(R-B)}{2\sqrt{(R-G)^2+(R-B)(G-B)}}\right] \quad R\neq B \quad or \quad G\neq B \tag{9-10}$$

$$S=1-\frac{3}{R+G+B}[\min(R,G,B)] \tag{9-11}$$

由式（9-10）直接计算出的 H 值在[0°,180°]之间，对应 $G\geqslant B$ 的情况，如图 9-10 所示。

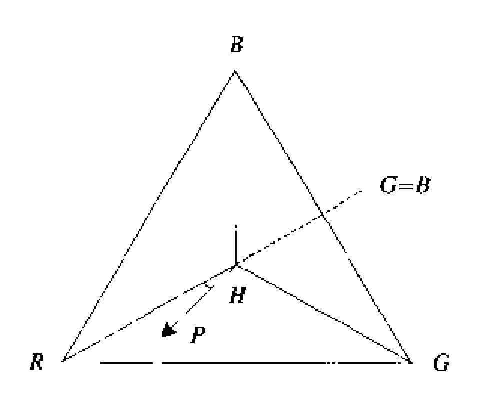
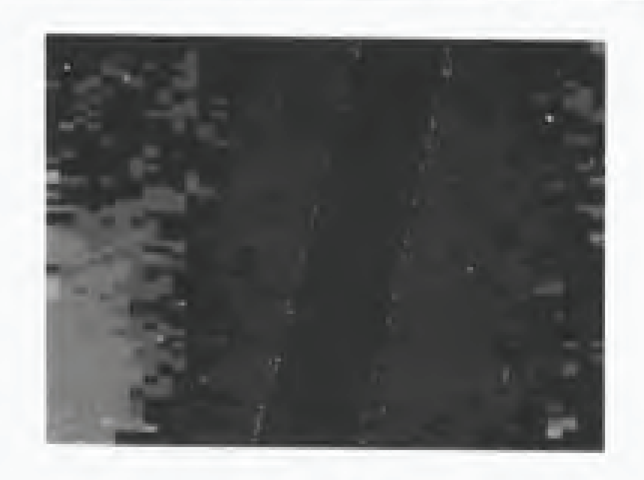
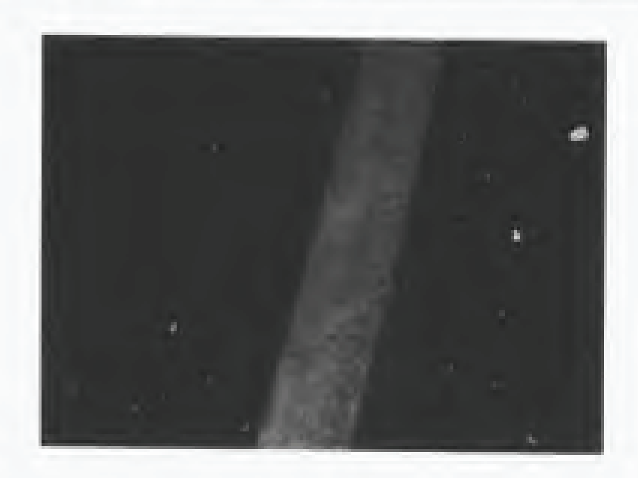


图 9-10　HIS 三角形图

在 $G<B$ 时，H 的值应该大于 180°。此时，令 H=360°-H，把 H 值转到[180°,360°]之间。而且当 H 值接近 0° 和 360° 时，表示的颜色应该是非常接近的，近似为红色，同一为 0 或 360°，否则该区域的颜色将明显地被分为两类。图 9-6 地路径用 HSI 彩色空间特征识别如图 9-11(a)～(c)所示。



(a) H分量图



(b) S分量图



(c) H、S分量路径识别



(d) 路径形态学处理

图9-11

【例9-3】HSI彩色空间的AGV路径识别（MATLAB程序代码）

```
I=imread('z12.JPG');
[y,x,z]=size(I);
myI=double(I);
tic
%%%%%%%%%%%% RGB to HSI  %%%%%%%%
H=zeros(y,x);
S=H;
HS_I=H;
for i=1:x
   for j=1:y
       HS_I(j,i)=((myI(j,i,1)+myI(j,i,2)+myI(j,i,3))/3);  % I分量，读者实际应用时可以不做
       S(j,i)=1-3*min(myI(j,i,:))/(myI(j,i,1)+myI(j,i,2)+myI(j,i,3));
       if  ((myI(j,i,1)==myI(j,i,2))&(myI(j,i,2)==myI(j,i,3)))   % 三者相等，式(10)分母为0
         Hdegree=0;
```

```
        else
    Hdegree=acos(0.5*(2*myI(j,i,1)-myI(j,i,2)-myI(j,i,3))/((myI(j,i,1)-myI(j,i,
2))^2+(myI(j,i,1)-myI(j,i,3))*(myI(j,i,2)-myI(j,i,3)))^0.5);
        end
        if (myI(j,i,2)>=myI(j,i,3))
            H(j,i)=Hdegree/(2*pi);              % G>=B,H 在[0 π]范围内
        else
            H(j,i)=(2*pi-Hdegree)/(2*pi);      % G<B,H 在[0 2π]范围内
        end
        if (H(j,i)>=0.9)
            H(j,i)=0;                           % H在0和2π附近应同属于一种颜色分量
        end
    end
end
t=toc
%%%%%%%%%%%%%%% Slect the White and Black %%%%%%%%%%%%%%%%%%%%%%
HS_I=uint8(HS_I);
H=255*H;
H=uint8(H);
S=255*S;
S=uint8(S);
I0=zeros(y,x);
for i=1:x
    for j=1:y

        if (((H(j,i)<=35)&(S(j,i)>=80)))
            I0(j,i)=1;
        else
            I0(j,i)=0;
        end
    end
end
SE=strel('rectangle', [12 8]);
IM=imclose(I0,SE);
figure(1);
imshow(I);
figure(3);
imshow(H);
figure,
imshow(S);
figure,
imshow(HS_I);
figure,imshow(I0);
figure,imshow(IM);
```

程序结果如图 9-11(a)~(d)所示。注意，这个程序运行时需要较长的时间，读者耐心等待。当然，在实际应用中，算法需要考虑在保证精度条件下如何提高实时性。

9.3.4 路径定位与方向偏差测量

路径图像识别的最终目的是为了获取AGV自导引车运行时车体与路径之间的相对位置偏差，以便控制器进行实时控制，使小车按照期望路径行驶。为了方便计算和分析，取车体纵轴线与路径中线的侧向偏差 e 和方向偏差 α 来表示，如图 9-12 所示。因此，必须从识别出的路径中检测出路径中心线，并以此定位中心线。

在 AGV 运行过程中，实际的导引路径基本上是由直线再加少量一定曲率半径的弧线（转弯处）组成的。由于 CCD 摄像头安装位置距离地面相对较近，拍摄到的图像是近视野图像，因此，可以将图像中的路径都近似看作直线来处理，在保证准确性的前提下大大减少计算量。

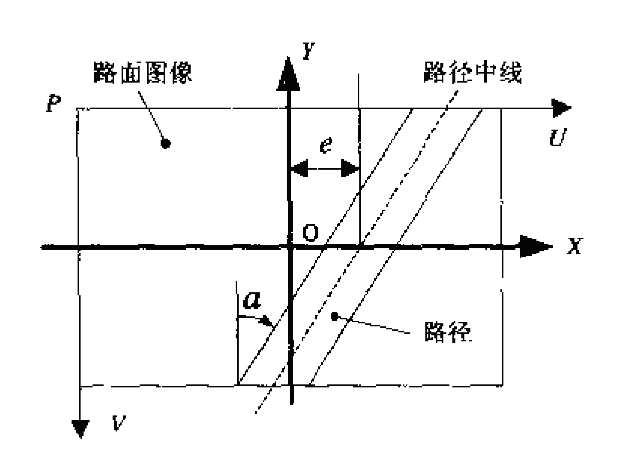


图 9-12 偏差示意图

路径定位通过路径中心线的位置检测实现。获取偏差时，对二值图像进行横向行扫描，分别找到各行上路径的左右边缘位置。

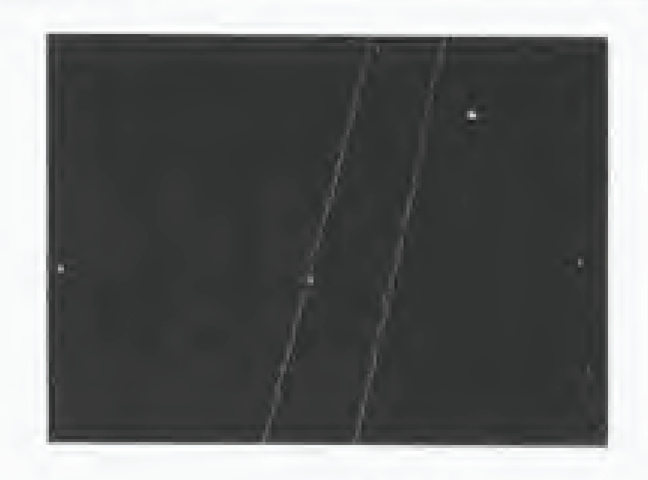
设其在图 9-12 所示的计算机坐标系 UPV（坐标原点 P 在图像的左上角）中的坐标分别为 (u_1, v_1)，(u_2, v_2)，则该行上路径中线上点的坐标（u_0，v_0）(u_0, v_0)为：

$$\begin{cases} u_0 = \dfrac{u_1 + u_2}{2} \\ v_0 = \dfrac{v_1 + v_2}{2} \end{cases} \tag{9-12}$$

图 9-5 和图 9-6 的路径图像识别结果均为二值图像（路面背景为黑色，灰度为 0；路径为白，灰度为 255），如图 9-13（a）所示。用数学形态学的“腐蚀”运算检测边缘，如图 9-13(b)所示，然后利用（9-12）式检测路径的中心线如图 9-13(c)所示。



(a) 路径的二值图像



(b) 路径的边缘



(c) 路径的中心线

图 9-13

在计算机处理时，将（9-12）式简化为水平方向逐行扫描，求取每一行路径图像的中心像素点的 U 方向的坐标值，减少计算量。虽然，用 3×3 结构的"腐蚀"运算，检测到的路径边缘为单个像素，但是水平方向（U 方向）的像素点并不会限于仅 2 个像素点。由于，路径的边缘较粗造，有时存在一段微小的水平直线，因此，在实际编程时，扫描检测最左和最右的路径像素点。

【例 9-4】路径中心线的定位（MATLAB 程序代码）

```
I=imread('z12R.jpg');
I=im2bw(I,0.6);
SE1=strel('square', 3);
R=imerode(I,SE1);                  % "腐蚀"运算
Edge_Road=I-R;                     % 边缘检测
[y,x]=size(I);
Px=zeros(y,1);
CenterR=zeros(y,x);
for j=1:y                          % 路径中线检测
   i=1;
```

```
    while((Edge_Road(j,i)~=1)&&(i<x))
          i=i+1;
    end
    if (i<x)
       P1=i;                    % 路径左边缘位置
    end
    i=x;
    while((Edge_Road(j,i)~=1)&&(i>1))
          i=i-1;
    end
    if (i>1)
       P2=i;                    % 路径右边缘位置
    end
    Px(j)=round((P1+P2)/2);  % 路径中线位置
    CenterR(j,Px(j))=1;
end
figure,imshow(R);
figure,imshow(Edge_Road);
figure,imshow(CenterR);
```

程序结果如图 9-13(a)~(c)所示。

1. 基于边缘的偏差提取

由于所得到的路径中心线有明显的波，可以用最小二乘法对求得的(u_0,v_0)进行线性拟合，得到路径中线的方程为：

$$au+bv+c=0 \tag{9-13}$$

将其转换到图像坐标系 XOY（坐标原点 O 在图像的中心）中，其坐标变换关系为：

$$\begin{cases} x=u-\dfrac{m-1}{2} \\ y=-(v-\dfrac{n-1}{2}) \end{cases} \tag{9-14}$$

式中，m，n 分别表示整幅图像的列数和行数。

将式（9-13）代入式（9-14）可得路径中线在图像坐标系 XOY 中的方程为：

$$ax-by+(\frac{m-1}{2}a+\frac{n-1}{2}b+c)=0 \tag{9-15}$$

令 $d=\dfrac{m-1}{2}a+\dfrac{n-1}{2}b+c$，则可由式（3－27）可求得侧向偏差 e 和方向偏差α:

$$e=-d/a \tag{9-16}$$

$$\alpha=\begin{cases}\dfrac{\pi}{2}-\left|\arctan(a/b)\right| & \dfrac{a}{b}>0\\ -[\dfrac{\pi}{2}-\left|\arctan(a/b)\right|] & \dfrac{a}{b}<0\end{cases} \tag{9-17}$$

2. Radon 变换的直线检测

路径的中心线波动也可以用 Radon 变换代替最小二乘法来消除。

Radon 变换是计算机图像在某一指定角度射线方向上的投影变换方法。Radon 变换与一种常用的计算机视觉操作 Hough 变换有着密切的关系。二维函数 $f(x,y)$的投影是其在确定方向上的线积分。例如，$f(x,y)$在垂直方向上的二维线积分就是 $f(x,y)$在 x 轴上的投影；$f(x,y)$在水平方向上的二维线积分就是 $f(x,y)$在 y 轴上的投影。图 9-14 说明了一个简单二维函数的水平和垂直投影。

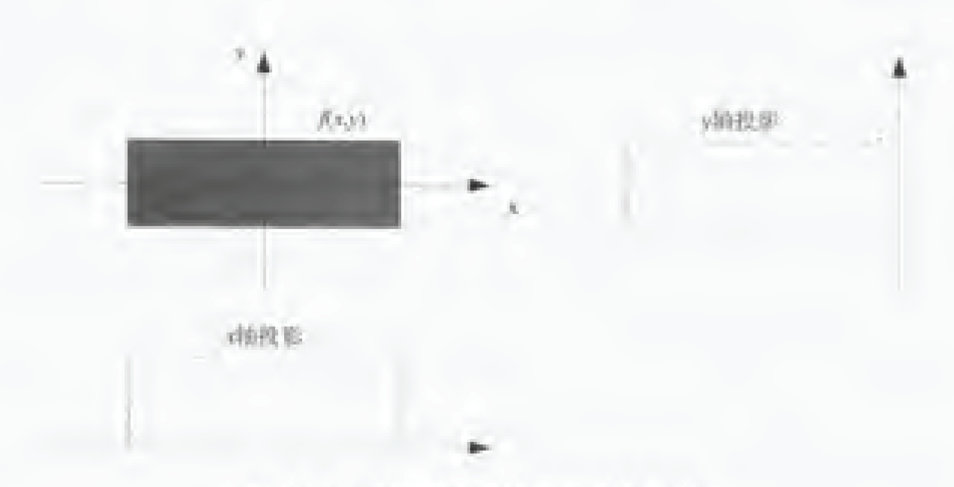


图 9-14　函数的水平投影和垂直投影

可以沿任意角度θ 计算函数的投影，也就是说，沿任意角度都存在函数的 Radon 变换。图 9-15 说明了 Radon 变换的几何原理。

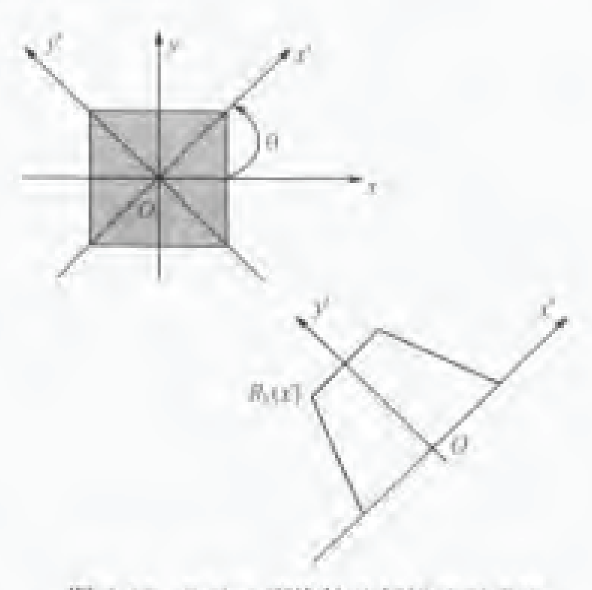


图 9-15　Radon 变换的几何原理示意图

通常情况下，$f(x,y)$的 Radon 变换是一个平行于 y 轴的线积分：

$$R_\theta(x') = \int f(x'\cos\theta - y'\sin\theta, x'\sin\theta - y'\cos\theta)dy' \tag{9-18}$$

式中，$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

Radon 变换检测的直线对是应于在 Radon 变换结果中坐标(θ,x')处存在峰值，根据θ 和 x' 的值确定一条直线。

对检测到的路径中心线进行 Radon 变换，结果如图 9-16 所示。图中最亮的点在θ=166.5°，x'=−36.5 处。

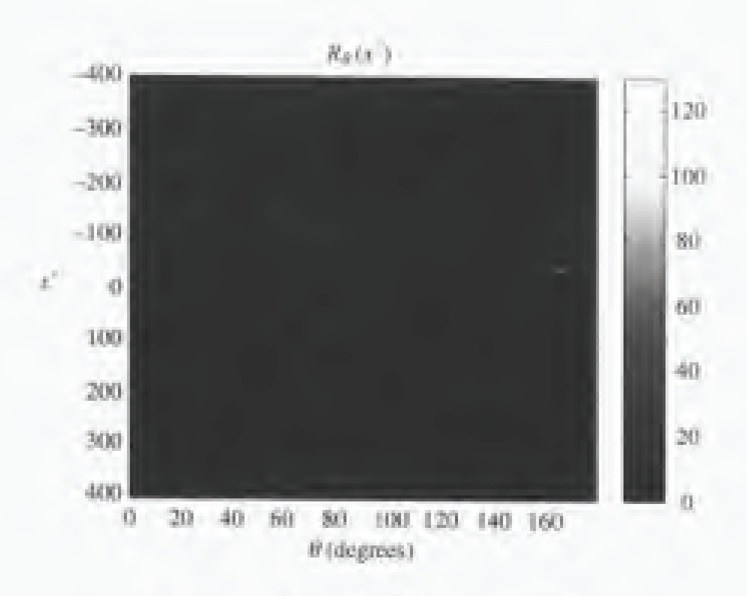


图 9-16　路径中心线的 Radon 变换

【例 9-5】Radon 变换的 AGV 路径偏差检测（MATLAB 程序代码）

```
theta = 0:179;
[R,xp] = radon(CenterR,theta);
figure, imagesc(theta, xp, R); colormap(hot);
xlabel('\theta (degrees)'); ylabel('X\prime');
title('R_{\theta} (X\prime)');
colorbar
```

程序结果如图 9-16 所示。注意，该程序在执行完例 9-5 的基础上运行。

第10章　图像技术在自动检测中的应用

生产过程中，为了排除次品和查明与故障有关的零件隐患等，要做产品的外观检查。以往多采用目测法，检测因人而异，容易漏检。利用图像技术研制了能在一定标准条件下，自动地检查每个零件的装置及识别系统。

零件的外观检查主要是发现产品或部件表面上的伤痕、变形或形状不良等。自动检查是图像技术应用的一个重要方面，本章主要介绍机械零件尺寸、机械振动和表面机加工质量的自动检测。

10.1　机械零件尺寸的自动检测

随着现代科学技术的发展，物体表面轮廓的三维测量已成为现代测试技术的一个重要分支。作为物体三维轮廓测量方法中，常用的有三坐标测量法、干涉测量法、莫尔等高线法、相位测量法等。基于非接触式在线测量，CCD图像测量方法较其他方法有其自身优势，测量效率高，对那些薄壁、软质易变形以及有化学污染一类的加工零件能实施非接触在线快速测量。

电耦合器件CCD由于具有自扫描、光电灵敏度高、几何尺寸精确、敏感单元尺寸小等一系列优点，因而被广泛用于工业非接触测量领域中。在精密定位、尺寸测量等方面，光敏面上均表现为空间光强分布的形式，经过数据采集和处理获得被测图像有关信息。随着计算机技术的普及，采用CCD器件和计算机数据采集、光学成像，机械运动组成的图像测量系统，能实时地对被测图像进行快速采样、存储和数据处理，实现光机电一体化设计。本节介绍对5000像元的线阵CCD摄像机及其图像卡对异型回转体轮廓的三维尺寸非接触测量进行研究。

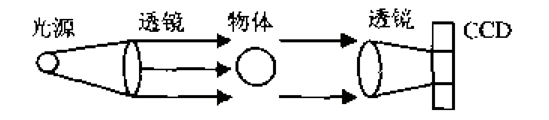


图10-1　CCD图像测量原理

1. 测量原理与方法

CCD图像测量原理如图10-1所示。当入射光在CCD像元上成像时，入射光被CCD像元吸收并产生相应数量的光生电荷，在光积分期间，光生电荷被积累并存储在彼此隔离的像元的势阱中，每个像元势阱中所积累的信号电荷数与照射在该像元面上的平均照度和光积分时间的乘积成正比，在电荷转移期间，光生电荷依次转移至输出区，通过复位脉冲的控制，在输出级形成视频信号，每次积分的输出波形代表目标光图像在CCD采样方向的瞬态强度的空间分布。由CCD输出的模拟信号是一个个脉冲信号，这些随时间变化的脉冲信号与随空间分布的光敏像元一一对应，通过物距、像距关系及CCD输出脉冲量的计算来检测零件尺寸。

针对异型回转体零件轮廓尺寸的检测进行研究，工件尺寸为R=100mm到R=120mm的变

化范围的异形回转体零件，考虑零件尺寸较大，如采用全视场的接收工件图像，CCD 采集到的是一个缩小倍率很大的像，这将引入较大误差。图 10-2 所示的一种测量方案，利用对工件轮廓边缘进行局部检测，将提高图像测量系统的分辨率和精度，有效地满足了测量精度的要求，图像检测装置如图 10-2 所示。

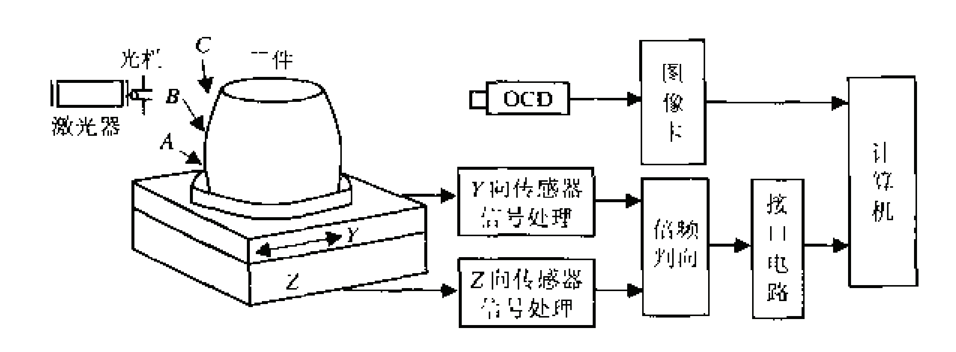
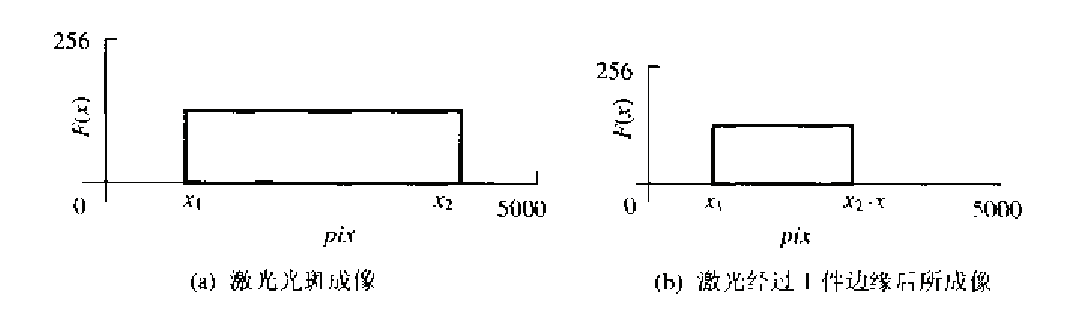


图 10-2　异形回转体外轮廓图像测量系统

这套测量系统包括：高精度回转工作台、光电轴角编码器、二维机械传动装置、两坐标直线位移传感器、PC 计算机、线阵 CCD（5000 线）、图像采集卡、接口卡、HeNe 激光器。

整个测量系统工作如下：设置一个沿 z、y 方向二维运动数显工作台，其位移分辨率为 0.5μm，由步进电机驱动使其完成沿水平方向和垂直方向的运动，二维工作台上安装一回转精度为 0.02(″)的高精度气浮回转台，可绕 z 轴转动，工件安装在回转工作台上，这样就可完成工件沿 z、y 方向的运动及绕 z 轴的转动，HeNe 激光器及 CCD 分别置于工作台两侧，使激光光斑成像于 CCD 器件的中央象元区间，此时的 CCD 输出信号如图 10-3(a)所示，当移动工件轮廓边缘 A 点于激光光斑处，使其边缘轮廓成像在 CCD 上时，其输出信号将改变为如图 10-3(b)所示，图中横坐标 pix 为 CCD 像元数，纵坐标为 CCD 采集到的信号灰度值。

由计算机记录光栅位移传感器当前点 $A(x,y)$的位移量以及 CCD 象元输出量，得到该点的位置，工件绕 z 轴旋转，由光电编码器控制按等间隔采样，这样工件同一剖面的实际轮廓尺寸误差值将直接反映到 CCD 像元上，通过 CCD 象元的输出量及物距、象距等关系便可计算出该工件 A 点剖面的二维实际尺寸，当移动工件 B点于 CCD 视场内，再由位移传感器记录下 $B(x,y)$点的位置量，旋转工件一周，计算出 B 点剖面的实际轮廓尺寸。重复以此，便可测量出工件三维轮廓的实际尺寸。试验过程中应用标定 CCD 尺寸当量的方法，可消除部分系统误差，达到较高的测量精度。



(a) 激光光斑成像　　(b) 激光经过工件边缘后所成像

图 10-3

2. 边缘轮廓检测及数据处理

被测对象经过 CCD 获得的原始测量数据往往带有高频噪声，微分的作用相当于一个高通

滤波器，因此对原始数据直接进行微分通常是不适用的，在边缘检测的数据处理中，采用前面图像分割讨论过的 Marr 提出的将高斯滤波和拉普拉斯边缘检测结合在一起，形成的 LoG（Laplacian-Gauss）算子检测轮廓边缘。并通过 CCD 像元个数的计数检测出轮廓尺寸。为了获取高精度的测量结果，测量前应用标准样块零件对系统进行标定。

10.2 机械振动幅值特征的图像测量

在传统的随机振动测量方法中，振动位移的拾取主要采用接触（利用机械式位移变换装置）或非接触（采用电容式、电感式、涡流式和霍尔效应传感器）方式，因而被测对象必须与传感器有机械或电磁耦合，从而不可避免地造成对被测对象的附加影响。然而，利用 CCD 图像传感器的线性时间积分成像特性，只需摄取被测点在静态下的图像及其在随机振动状态下的一组连续的时间平均成像即可确定该点的幅值统计特征。鉴于各态历经随机振动在工程中经常遇到，本文研究时间平均成像方法在各态历经随机振动测量中的应用。

10.2.1 CCD 线性时间积分成像原理

CCD 图像传感器是由按照一定规律排列的 MOS（金属-氧化物-半导体）电容器阵列组成的位移寄存器，这种电容能存储电荷。在光照情况下，CCD 的感光单元存储的电荷正比于曝光量 E（曝光量 E 等于辐射光强 I 与曝光时间$\triangle t$ 的乘积）。图 10-4 为线阵 CCD 的光电转换特性，从中可以看出它有一线性工作区，而曝光量达到饱和曝光量 E_S 后，它所存储的电荷达到最大值 Q_S，并不再继续增加。CCD 存储的电荷经图像采集系统转移输出并量化成为数字灰度图像。

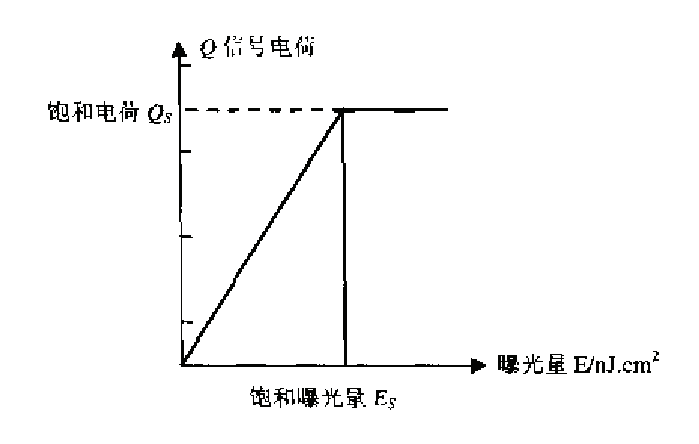


图 10-4 CCD 光电转换特性

10.2.2 测量系统的组成

如图 10-5 所示，测量系统由光学系统、线阵 CCD、图像采集系统和液晶显示屏组成。在被测物体上选择一待测点，激光光源照射在待测点上，并保证被测物体上除待测点外其余各点及背景均无光照辐射，被测点坐标沿纵轴作随机振动，坐标原点与被测点的平衡位置重合。待

测点上的光照经光学镜头后，在像平面上聚焦成一光点。采用线阵 CCD 记录被测点的图像，摄像系统的像平面平行于振动平面。显然，当被测点沿纵轴作随机振动时，光点在像平面上的投影也随之作同步振动。

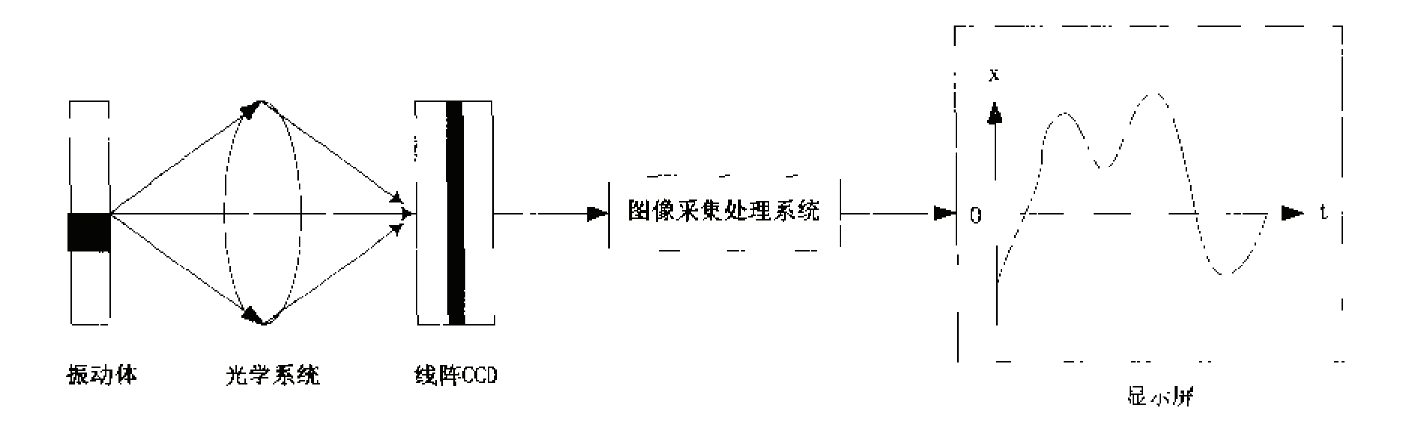


图 10-5 振动测量系统的组成

设光学系统的放大倍数为 m，被测点在任一时刻 t 的实际振动振幅为 $x(t)$，则像平面上光点的振幅为 $z(t)$，即：

$$x(t) = z(t)/m \tag{10-1}$$

10.2.3 被测点时间平均成像与振幅特征之间的关系

由于 CCD 具有时间积分成像效应，当被测点沿纵轴作横向随机振动时，它在像平面上的时间平均成像形成一条平行于纵轴的线状光带。由于振动过程中，其运动速度不断变化，而 CCD 像素点的大小固定，因此，在不同像素点上，光点照射的时间也不相同，根据 CCD 积分成像效应，各像素点的灰度值也不相同。越靠近平衡点的光点照射时间越短，灰度值越小。但随机振动，使得各像素点的灰度值呈现随机波动。线阵 CCD 只对某种波长（本系统的激光光源）的光敏感，因此 CCD 的灰度值不会被环境光所干扰。

设光带沿纵轴的灰度分布为 $g(z)$。被测光点作各态历经随机振动，CCD 成像曝光时间（相当于样本长度）为 T，则被测点的振动副值特征与其时间平均成像之间的关系如下。根据定义，各态历经随机振动的均值 μ_x 等于样本函数 $x(t)$ 的时间平均值，即：

$$u_x = \lim_{T\to\infty}\frac{1}{T}\int_0^T x(t)dt = \lim_{T\to\infty}[\frac{1}{T}\lim_{\substack{\Delta t\to 0\\ N\to\infty}}\sum_{i=1}^{N}x(t_i)\Delta t_i] \tag{10-2}$$

设样本函数 $x(t)$ 在时间段 T 内的动态范围为 $[x_a,x_b]$，将上式按照另外一种方法计算：将 $[x_a,x_b]$ 分成 M 段，每一段的宽度为 Δx，设被测点的位置落在第 j 段 $[x_j,x_j+\Delta x]$ 内的总时间为 Δt_j，则式（10-2）可以改写为：

$$u_x = \lim_{T\to\infty}\frac{1}{T}\int_0^T x(t)dt = \lim_{T\to\infty}[\frac{1}{T}\lim_{\substack{\Delta t\to 0\\ M\to\infty}}\sum_{i=1}^{M}x(t_j)\Delta_j t] \tag{10-3}$$

从式（10-1）可知，被测点在像平面上的投影对应的动态范围为 $[z_a,z_b]$，Δx 则对应于像平

面上的 $\Delta z=m\Delta x$；被测点的位置 x_j 对应于它在像平面上的投影位置 $z_j=mx_j$，而 $[x_j,x_j+\Delta x]$ 则对应于 $[z_j,z_j+\Delta z]=[mx_j,mx_j+m\Delta x]$。根据 CCD 线性时间积分成像特性，CCD 在像平面上所记录的图像中任一点的灰度与该点所受的曝光量成正比，则 Δt_j 与被测点在振动状态下的时间平均成像沿纵轴的灰度分布 $g(z)$ 之间的关系为

$$\int_{z_j}^{z_j+\Delta z} g(z)dz = kI\Delta_j t \tag{10-4}$$

其中，I 为被测点辐射在像平面上的光强，k 为 CCD 的光电转换系数。

同理，曝光时间 T 与灰度分布 $g(z)$ 之间的关系为

$$\int_{z_a}^{z_b} g(z)dz = kIT \tag{10-5}$$

分别从（10-4）和（10-5）解出 Δt_j 和 T，将他们代入式（10-3）并整理得

$$u_x = \lim_{T\to\infty}\{\frac{Ik}{\int_{z_{aj}}^{z_b} g(z)dz}\lim_{\substack{\Delta x\to 0\\ M\to\infty}}\sum_{j=1}^{M}[\frac{1}{mIk}z\int_{z_j}^{z_j+\Delta z} g(z)dz]\} = \lim_{T\to\infty}\frac{\int_{z_{aj}}^{z_b} zg(z)dz}{m\int_{z_{aj}}^{z_b} g(z)dz} \tag{10-6}$$

因此，可以从被从线阵 CCD 摄取的图像中确定各态历经随机振动的幅值统计特性。同理，可以得到均方值，方差和概率密度函数等随机振动特性参数。

时间平均成像方法可以直接测量各态历经随机振动的幅值统计特征一位移动态范围、均值、均方值、方差和幅值概率密度曲线，并具有很高的精度。这种方法所需测量设备简单，无需同被测对象接触而且可以测量很宽的频率成份，具有方便快捷的优点。在测量时应保证所采集的每幅图像不出现曝光饱和，此外图像传感器应有很好的线性光电转换特性、较大的动态范围和灵敏度的空间均匀性。

10.3 钢球表面缺陷的自动检测与识别

作为成品钢球的验收程序，最后必须进行表面缺陷检验。传统的检测方法是将钢球置于显微镜下放大 100 倍，再与标准照片目视对照。这种人工检验方法，无疑需要大量人力，生产效率难以提高；检验人员长时间地观察显微镜，眼睛受到刺激容易疲劳并伴有疼痛感，产生漏检的机会将大大增加。

随着数字图像处理与模式识别技术的不断发展和经济效益要求的逐步提高，迫切需要有一种自动检测装置，正确有效地对钢球表面缺陷进行分类。利用数字图像处理手段，根据钢球表面缺陷区域的几何特征和纹理特征，定量地表征缺陷的性质，并且根据模式识别理论设计了分类器。识别结果将给出缺陷的种类，或“无缺陷”的评价。

10.3.1 系统的组成

图 10-6 为已开发的钢球表面缺陷检测系统的框图。本装置由自制钢球载物台、金相显微镜、工业摄像机、图像采集卡和计算机等 5 部分组成。载物台可使钢球任意旋转，以图像，自定义其扩展名为 RAW 格式的原图像文件并存盘，由开发的图像处理软件读取原文件，并进行

图像预处理，特征提取之后，进行模式识别，最后给出模式识别结果。

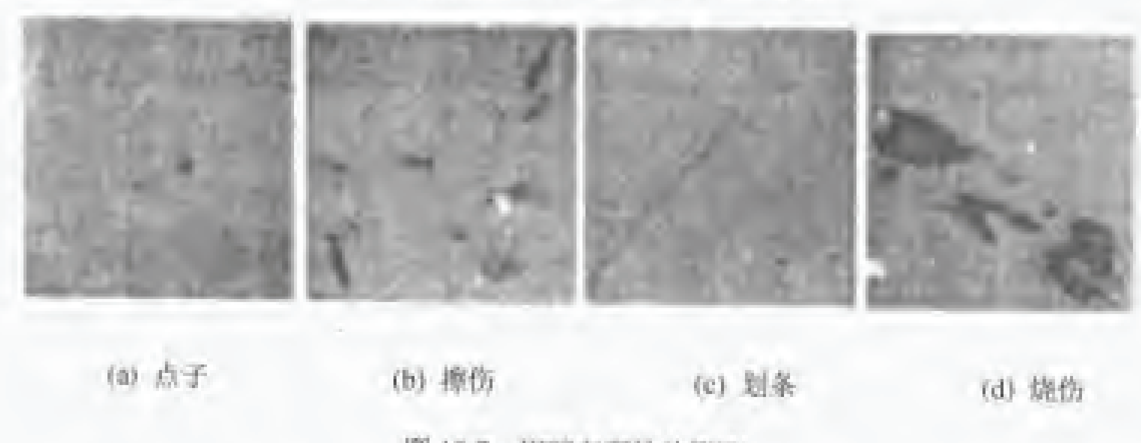


图 10-6　钢球表面缺陷检测系统的框图

10.3.2　图像预处理

1. 二值化

钢球表面缺陷图由连接在显微镜上的摄像机取出，记取在图像缓冲器上后，将灰度图像二值化，除去干扰，可进行缺陷区域的边界跟踪。二值化以下的处理由计算机进行阐述阈值的确定、排除干扰、缺陷区域边界的自动跟踪及缺陷区域图像的复原。

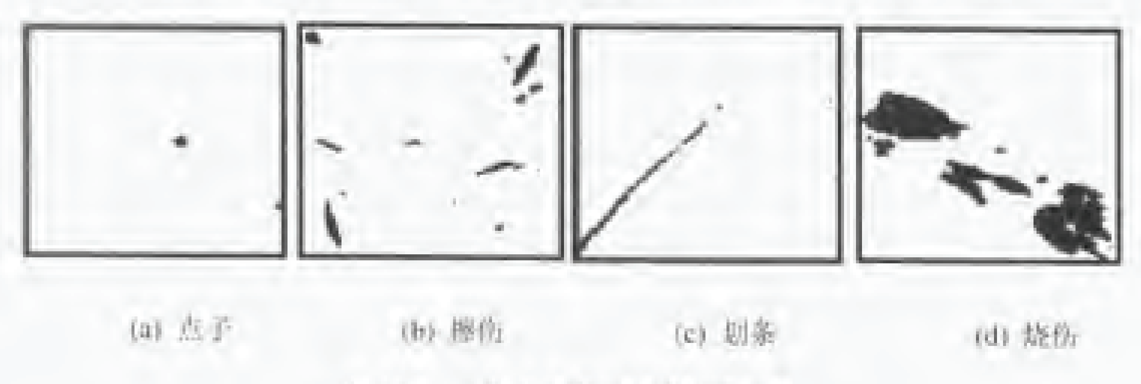


(a) 点子　　(b) 擦伤　　(c) 划条　　(d) 烧伤

图 10-7　钢球表面缺陷原图

钢球表面的缺陷如图 10-7 所示。为了检测出缺陷的边缘，进行图像分划（缺陷子图像的提取），应先把缺陷与背景分离开。灰度分割阈值 T 的选择相当重要。在钢球显微图像的直方图上不一定会得到理想的物体峰，多数情况下会呈现不规则形状，但由于背景图像分布往往满足高斯分布，会出现一个较为突出的背景峰，利用这一特点，提出了一种新的阈值计算方法——双窗口法。其算法处理顺序如下：①在摄像头拍摄的图像内设定包含缺陷部分的窗口 A，计算这个范围内的像素点灰度分布，可得原始图像的直方图，用加窗高斯函数对该直方图进行滤波。加窗高斯函数将加窗高斯函数的离散值同直方图函数的离散值进行卷积，就完成了滤波操

件。②设定一个包含有窗口 A 的窗口 B。同样求出像素点灰度分布，则可得出灰度分布 B。则灰度分布 A 与 B 的差，相当于背景部分的灰度成分；③在前项将窗口的灰度分布由 A 扩大至 B，无变化的灰度成分属于缺陷部分，可求出阈值 T。分割出二值灰度缺陷图像。



(a) 点子　　(b) 擦伤　　(c) 划条　　(d) 烧伤

图 10-8　二值灰度分割后的缺陷图像

2. 去除噪声

一般的钢球显微图像中，存在反射或因图像处理机器自身的误差，所以将原图像照旧处理则出现误差。利用前述的阈值 T，二值化后的图像中也存在受其影响的斑点。对这幅图像实行 AND 理论及 OR 理论处理，可得到已被排除干扰的图像。排除干扰的方法如图 10-9 所示，先将对象像素 CP 与其相邻的 4 个像素之间进行 AND 运算，作为新生的 CP 值。这个处理是关于二值图像白领域的缩小处理，由 4 个白像素包围的部分像素作为白像素的残留。之后，与缩小运算同样地在噪声处理对象的像素和相邻像素之间进行 OR 运算，扩张处理又返回最初的白领域。这两个处理，点状的干扰、二像素的线状的白领域作为干扰都被排除了。

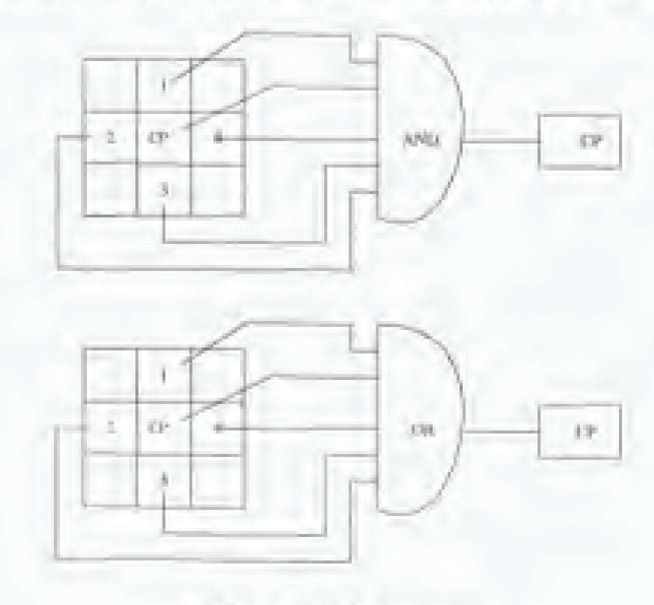


图 10-9　噪声滤去方法

3. 缺陷边界跟踪

将原图像二值化后，分为缺陷区域和背景区域，抽出缺陷区域的边界，计算缺陷区域的几何特征参数。缺陷边界跟踪步骤如下。

（1）抽出上侧边界点集合

将缺陷图像自上而下扫描，初次出现的白像素点作为上侧边界点，处理全体区域求出上侧

边界点的集合，上侧边界点相当于缺陷区域和背景的边界。

（2）缺陷边界曲线的确定

以上侧边界点集合的一点为开始点，追踪缺陷边界，这个边界通常是以移动方向的左侧为黑像素，右侧为白像素来搜索边界，哪个方向都不存在白像素时返回，进行方向变为 Y 继续用同样的步骤处理，直至回到开始点为止继续追踪，则可得缺陷区域边界的闭曲线。

4. 缺陷区域图像的复原

为了研究缺陷区域灰度变化规律，将二值图像的缺陷区域用原有灰度级来表示，以提取纹理特征。

$$g(x,y)=\begin{cases}G_m(x,y) & G_m>T\\ L_b & G_m\leqslant T\end{cases} \tag{10-7}$$

10.3.3 缺陷特征提取

由本系统采集的钢球表面缺陷图像见图 10-7。通过对大量显微图像观测，钢球表面经常产生的缺陷包括点子、群点、擦伤、划条及烧伤等。点子、擦伤的几何特征是非常明显的，将此类缺陷称之为面型缺陷。而划条、凹坑、烧伤除用几何特征描述外，还必须考虑缺陷区域的纹理特征（区域的波动性），将此类缺陷称之为立体型缺陷。

显微图像经过处理后，可从中提取一些能反映缺陷性质而且相对比较稳定的特征，作为分类、识别和理解缺陷的依据。鉴于显微图像处理的现状，目前不可能建立各种缺陷的专家知识库，缺陷特征的选择主要依赖于缺陷的特征分析和实验分析结果。这里选择了 4 个特征，各特征的数学模型如下：

（1）缺陷的面积

由于不同缺陷的面积大小差异较大，为了简化计算，采用缺陷区域所包含的像素点个数来计算面积的大小：

$$S=\sum_{x=1}^{M}\sum_{y=1}^{N}G_m(x,y) \tag{10-8}$$

式中，M，N 表示缺陷数字图像的尺寸大小。

（2）缺陷短径和长径之比

缺陷短径 L_2 定义为与长径垂直的一组直线 O_1O_1，O_2O_2，O_3O_3……中，最大的直线长度，如图 10-10 所示。

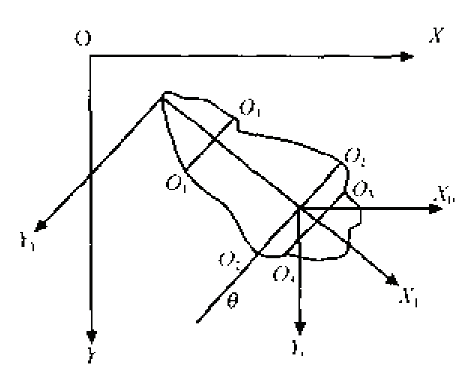


图 10-10　短径计算示意

长径 L_1 用边界上任两个点之间的最大距离来表示，设 $a_1(x_1,y_1)$和 $a_2(x_2,y_2)$是任意边界上的两点，则：

$$L_1 = \max[\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}] \quad (10\text{-}9)$$

（3）缺陷图像的纹理特征

对于立体型缺陷，光线反射缺陷区域的波动性存在较大的差别，这种差别表现为不同的灰度变化规律。为了客观反映缺陷区域内的灰度变化规律，选用灰度方差σ_e和熵σ_b共 2 个特征，定义如下：

$$\sigma_b^2 = \sum_{i=0}^{G-1}(b_i - \mu_b)^2 p(b_i) \quad (10\text{-}10)$$

$$\sigma_e = -\sum_{i=0}^{G-1} p(b_i)\log_2 p(b_i) \quad (10\text{-}11)$$

$$\mu_b = \sum_{i=0}^{G-1} b_i p(b_i) \quad (10\text{-}12)$$

式中，$p(b_i)$为缺陷区域内各像素点灰度值为 b_i 的概率，G 为灰度级。

10.3.4 缺陷识别

鉴于钢球表面缺陷图像各特征间有一定的相关性，为降低识别算法的复杂性，特征识别时采用阶层识别的方法。利用二叉树线性分类器，逐层选用上一节中的特征，将各类缺陷一一检出。选择特征的准则是同种特征值相差最明显的，保证特征具有较大的相互独立性。特征阈值分类阈值由实验分析“钢球表面质量标准照片图册（ZQ8-83 和 ZQ81-85）”和特征分析结果得到。作为典型的分析示例，通过图像预处理计算其各缺陷有关的特征。在分类过程中、由于短径与长径比这一特征的明显优势，首先将划条和擦伤缺陷从几种缺陷中分离出来，再由 L_2/L_1 的 T_2 阈值分离划条和擦伤，然后根据面积特征使点子和烧伤得到识别。其分类二叉树表示如图 10-11 所示，T_1、T_2 和 T_3 分别为相应的分类判别阈值缺陷，通过实验分析得到，也可由样本学习得到。其中第二步分类确定划条和擦伤时必须经过纹理特征方差σ_e和熵σ_b检验。

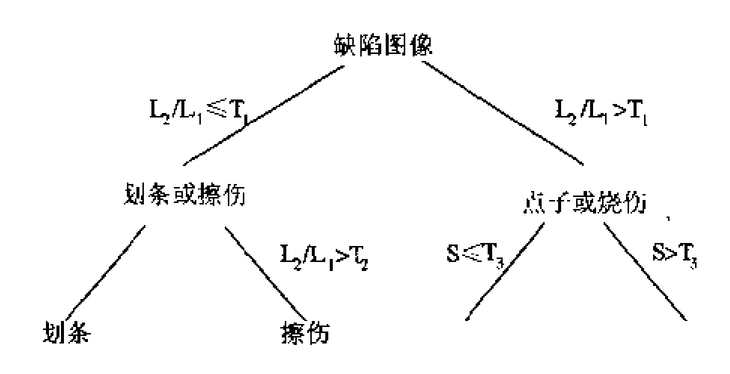


图 10-11 钢球缺陷分类二叉树表示

基于图像处理和模式识别理论开发的钢球表面缺陷自动检测系统，有效地完成了钢球表面缺陷的识别。这个系统是由工业摄像机捕捉钢球表面缺陷图像，自动检出缺陷区域的轮廓，算出缺陷特征参数。为正确求出缺陷区域的轮廓，使用了如下的图像处理算法：

（1）使用双窗口法，确定二值化的阈值；

（2）采用领域缩小和扩张的方法消除图像的干扰；

（3）由边界跟踪抽出缺陷轮廓线；

（4）使用缺陷子图像复原法，求出缺陷区域的纹理特征。

根据这些图像处理算法，采用二叉树线性分类器，对钢球表面缺陷图像进行了正确分类。

第 11 章　基于神经网络的文字识别系统

11.1　系统简介

该系统是为了辨认识别图像中的数字而设计的，它通过对图片的一系列处理，最后识别得出图片中显示的数字。系统既可以单独使用，也可以把它作为一个识别系统的软件核心应用到车牌识别等系统中去。

11.2　系统的基本技术要求

下面是系统具体要达到的基本技术要求

（1）输入图片中可以含有多个数字；
（2）数字的识别准确率大于 90%；
（3）每张图片的处理时间（识别时间）不能大于 1s；
（4）对图片噪声具有较强的适应性；
（5）系统要能长时间无故障运行；
（6）系统的操作要求简单。

11.3　系统中的关键技术

在本系统中用到了好多图像处理中的相关技术：比如灰度化、二值化、图像内容自动调整、去离散点、图像的缩放、细化、曲线平滑、曲线去枝桠操作等，最后还使用了神经网络对提取到的数字信息进行分析判断。

11.4　系统的软硬件平台

11.4.1　系统的硬件平台

因为系统运行的过程当中，主要进行的都是图像处理，在这个过程当中要进行大量的数据处理，所以处理器和内存要求比较高，CPU 要求主频在 600Hz 及其以上，内存在 128MB 及其以上。

11.4.2　系统的软件平台

系统可以运行于 Windows 98、Windows 2000 或者 Windows XP 操作系统下。程序调试时，需要使用 Microsoft Visual C++ 6.0（SP6）。

11.5　系统实现

系统在实现的过程当中，先分解成两个大块，就是图像预处理模块和数字识别模块。其中图像像预处理块在对图像进行了一系列变换后把最后提取到的数字字符提交给数字识别模块，然后进行识别并给处结果。在这里用到了很多先进的图像预处理技术及神经网络技术.

11.5.1　系统流程图

本系统总的流程为“图像预处理”→“神经网络文字识别”。

其中图像预处理的流程如图 11-1 所示。

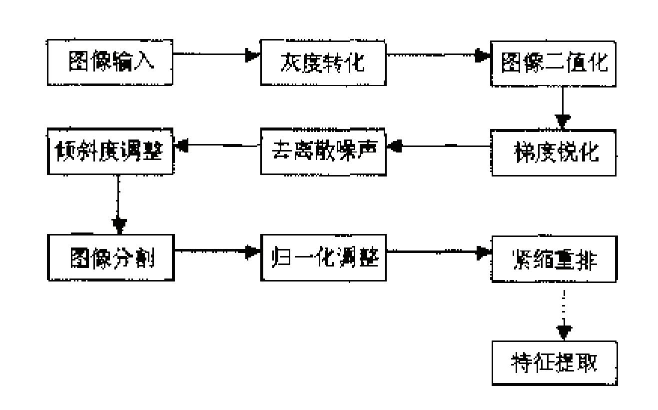


图 11-1　图像预处理流程图

神经网络数字识别的具体流程如图 11-2 所示。



图 11-2　神经网络数字识别流程图

11.5.2　程序实现

整个系统的程序实现分为图像预处理和神经网络识别两大模块。在图像预处理的过程当中，采用了许多图像处理的技术，最后把每个数字的特征提取出来。这些技术包括图像数据读取、图像的灰度化、二值化、图像的调整、离散噪声点的去除、字符的切分、图像的缩放、字符的细化、字符的平滑、图像的求梯度等图像处理技术，最后是数字字符特征的提取。其结果再利用神经网络（这里选用 BP 网络）进行字符识别。

利用神经网络进行字符识别的过程主要包括网络的训练、数据的读取、字符的判定、结果的输出。下面按照程序执行的顺序介绍整个程序并逐一分析每一步的关键代码。最后简要讲述一下程序的使用方法、注意事项以及本章小结。

1. 图像数据的读取、保存与屏幕显示等基本函数

要进行图像分析和处理首先就要得到图像的数据，这些数据包括图像的宽、高、每个像素点的颜色值。因为每种文件都有它自己的存放格式，下面简单介绍 BMP 文件的存放格式。本程序采用的输入图像即为 BMP 位图。

BMP（bitmap 的缩写）文件格式是 Windows 本身可以直接提供读取支持的位图文件格式。一个 BMP 格式的文件通常有 BMP 的扩展名，但有一些是以 rle 为扩展名的，rle 的意思是行程长度编码（run length encoding）。这样的文件意味着其使用的数据压缩方法是 BMP 格式文件支持的两种 rle 方法中的一种。BMP 文件可用每像素 1、4、8、16 或 24 位来编码颜色信息，这个位数称作图像的颜色深度，它决定了图像所含的最大颜色数。一幅 1-bpp（位每像素，bit per pixel）的图像只能有两种颜色。而一幅 24-bpp 的图像可以有超过 16 兆种不同的颜色。

一个典型 BMP 文件的结构。以 256 色也就是 8-bpp 为例，文件被分成 4 个主要的部分：一个位图文件头，一个位图信息头，一个颜色表（又称为色表）和位图数据本身。

（1）位图文件头

位图文件头包含关于这个文件的信息。如从哪里开始是位图数据的定位信息、文件大小等等。以下是位图文件头结构的定义：

```
typedef struct tagBITMAPFILEHEADER
{ // bmfh
WORD  bfType;
DWORD  bfSize;
WORD  bfReserved1;
WORD  bfReserved2;
DWORD  bfOffBits;
} BITMAPFILEHEADER;
```

其中的 bfType 值应该是“BM”（0x4d42），标志该文件是位图文件。bfSize 的值是位图文件的大小。

（2）位图信息头

位图信息头包含了单个像素所用字节数以及描述颜色的格式，此外还包括位图的宽度、高度、目标设备的位平面数、图像的压缩格式。

以下是位图信息头结构的定义：

```
typedef struct tagBITMAPINFOHEADER{ // bmih
  DWORD biSize;
  LONG  biWidth;
  LONG  biHeight;
  WORD  biPlanes;
  WORD  biBitCount
  DWORD biCompression;
  DWORD biSizeImage;
  LONG  biXPelsPerMeter;
  LONG  biYPelsPerMeter;
```

```
    DWORD biClrUsed;
    DWORD biClrImportant;
} BITMAPINFOHEADER;
```

下面是对结构体当中各个成员的说明：

biSize——结构 BITMAPINFOHEADER 的字节数，即 sizeof(BITMAPINFOHEADER)；

biWidth——以像素为单位的图像宽度；

biHeight——以像素为单位的图像长度；

biplanes——目标设备的位平面数；

biBitCount——每个像素的位数；

biCompression——图像的压缩格式（这个值几乎总是为 0）；

hiSizeImage——以字节为单位的图像数据的大小（对 BI_RGB 压缩方式而言）；

biXPelsPermeter——水平方向上的每米的像素个数；

biYPelsPerMeter——垂直方向上的每米的像素个数；

biClrused——调色板中实际使用的颜色数，这个值通常为 0，表示使用 biBitCount 确定的全部颜色，例外是使用的颜色的数目小于制定的颜色深度的颜色数目的最大值；

biClrImportant——显示位图时的颜色数，这个值通常为 0，表示所有的颜色都是必需的。

对于其中的 biBitCount，分别有以下意义：

0——用在 JPEG 格式中；

1——单色图，调色板中含有两种颜色，也就是我们通常说的黑白图片；

4——16 色图；

8——256 色图，通常说的灰度图；

16——64K 图，一般没有调色板，图像数据中每两个字节表示一个像素，5 个或 6 个位表示一个 RGB 分量；

24——16M 真彩色图，一般没有调色板，图像数据中每 3 个字节表示一个像素，每个字节表示一个 RGB 分量；

32——4G 真彩色，一般没有调色板，每 4 个字节表示一个像素，相对 24 位真彩图而言，加入了一个透明度，即 RGBA 模式。

（3）颜色表

颜色表（文中简称色表）一般是针对 16 位以下的图像而设置的，对于 16 位和 16 位以上的图像，由于其位图像素数据中直接对对应像素的 RGB(A)颜色进行描述，因而省去了调色板。而对于 16 位以下的图像，由于其位图像素数据中记录的只是调色板索引值，因而需要根据这个索引到调色板去取得相应的 RGB(A)颜色。颜色表的作用就是创建调色板。对显示卡来说，如果它不能一次显示超过 256 种颜色，读取和显示 BMP 文件的程序能够把这些 RGB 值转换到显示卡的调色板来产生准确的颜色。

颜色表的颜色表项结构的定义如下：

```
typedef struct tagRGBQUAD { // rgbq
    BYTE  rgbBlue;
    BYTE  rgbGreen;
    BYTE  rgbRed;
    BYTE  rgbReserved;
} RGBQUAD;
```

其中需要注意的问题是，RGBQUAD 结构中的颜色顺序是 BGR，而不是平常的 RGB。

（4）位图数据

BMP 文件的位图数据格式依赖于编码每个像素颜色所用的位数。对于一个 256 色的图像来说，每个像素占用文件中位图数据部分的一个字节。像素的值不是 RGB 颜色值，而是文件中色表的一个索引，所以在色表中如果第一个 R/G/B 值是 255/0/0，那么像素值为 0 表示它是鲜红色，像素值按从左到右的顺序存储，通常从最后一行开始。所以在一个 256 色的文件中，位图数据中第一个字节就是图像左下角的像素的颜色索引，第二个就是它右边的那个像素的颜色索引。如果位图数据中每行的字节数是奇数，就要在每行都加一个附加的字节来调整位图数据边界为 16 位的整数倍。

并不是所有的 BMP 文件结构都像表中所列的那样，例如 16 和 24-bpp 文件就没有色表，像素值直接表示 RGB 值，另外文件私有部分的内部存储格式也是可以变化的。例如，在 16 和 256 色 BMP 文件中的位图数据采用 rle 算法来压缩，这种算法用颜色加像素个数来取代一串颜色相同的序列，而且，Windows 还支持 OS/2 下的 BMP 文件，尽管它使用了不同的位图信息头和色表格式。为了减小运算的数据量，本程序使用 256 色 BMP 文件作为输入。

近年来，一种后缀名为 jpg 的位图凭借较高的压缩比和不俗的品质，并借助网络的优势而迅速成为图像/图形行业事实上的工业标准。本文没有采用 JPG 位图，但在这里也简要的介绍一下，以供读者参考。

JPG 是 24 位的图像文件格式，也是一种高效率的压缩格式，文件格式是 JPEG（联合图像专家组）标准的产物，该标准由 ISO 与 CCITT（国际电报电话咨询委员会）共同制定，是面向连续色调静止图像的一种压缩标准。其最初目的是使用 64kbit/s 的通信线路传输 720×576 分辨率压缩后的图像。通过损失极少的分辨率，可以将图像所需存储量减少至原大小的 10%。由于其高效的压缩效率和标准化要求，目前已广泛用于彩色传真、静止图像、电话会议、印刷及新闻图片的传送上。但那些被删除的资料无法在解压时还原，所以* .jpg 文件并不适合放大观看，输出成印刷品时品质也会受到影响。不过，普通用户不必担心，因为*.jpg 的压缩算法十分先进，它对图形图像的损失影响不是很大，一幅 16MB（24 位）的*.jpg/*.jpeg 图像看上去与照片没有多大差别，非专业人士甚至无法分辨。同样一幅画面，用*.jpg 格式储存的文件是其他类型图形文件的 1/10～1/20。一般情况下，*.jpg 文件只有几十 KB，而色彩数最高可达到 24 位，所以它被广泛运用在 Internet 上，以节约宝贵的网络传输资源。同样，为了在一张光盘上储存更多的图形图像，CD 出版商也乐意采用*.jpg 格式。目前，网上已经有了很多 jpg 图像的编解码的算法，限于篇幅，本文就不再赘述。如果要使用 jpg 格式的图像，那么请先用画图软件如 ACDSee、Photoshop 将其转化为 256 色 bmp 位图格式。另外，较常用到的图像格式还有 gif、tiff 和 png 等。由于本文的核心主要集中在图像的预处理和 BP 神经网络识别部分，就不在图像格式上耗费精力了。读者需要进一步了解图像格式的可以详细查阅相关资料。

在图像预处理部分的图像数据读取部分，作者使用了微软提供的一个图像函数库 dibapi.h 和 dibapi.cpp，里面已经含有一些基本的图像处理函数，作者在此就不再列举源代码，而紧紧将其接口加以描述，以使读者清晰每个函数的作用。同时作者自己对该库又加以扩充以满足本程序的需要。文中没有贴出而程序中又用到的代码都在本书的附带光盘中可以找到。

本小节所有的函数定义及声明位于 dibapi.h、dibapi.cpp 和 mydiblib.h 中。

首先来看 7 个图像数据读取/存储/创建以及图像基本信息获取函数。

图像数据读取/保存由函数 ReadDIBFile、SaveDIB、FindDIBBits、DIBWidth、DIBHeight、DIBNumColors 完成，其调用接口如下。

（1）ReadDIBFile 函数

函数原型：

```
HDIB  WINAPI  ReadDIBFile(CFile& file);
```

参数：

```
CFile& file            - 要读取得文件文件 CFile
```

返回值：

```
HDIB                   - 成功返回 DIB 的句柄，否则返回 NULL。
```

说明：该函数将指定的文件中的 DIB 对象读到指定的内存区域中。除文件头之外的内容都将被读入内存。HDIB 即此内存区域的指针

（2）SaveDIB 函数

函数原型：

```
BOOL  WINAPI  SaveDIB (HDIB hDib, CFile& file);
```

参数：

```
HDIB hDib              - 要保存的 DIB
CFile& file            - 保存文件 CFile
```

返回值：

```
BOOL                   - 成功返回 TRUE，否则返回 FALSE 或者 CFileException
```

说明：该函数将指定的 DIB 对象保存到指定的 CFile 中。该 CFile 由调用程序打开和关闭。

（3）FindDIBBits 函数

函数原型：

```
LPSTR     WINAPI  FindDIBBits (LPSTR lpbi);
```

参数：

```
LPSTR lpbi             - 指向 DIB 对象的指针
```

返回值：

```
LPSTR                  - 指向 DIB 图像像素起始位置
```

说明：该函数计算 DIB 中图像像素数据区的起始位置，并返回指向它的指针。

注意：LPSTR 指针为指向字符串的 32 位的指针。在对 256 色图像进行像素操作时，可以用(BYTE*)或(unsigned char*)强制将其转换为 8 位的指针

（4）DIBWidth 函数

函数原型：

```
DWORD WINAPI DIBWidth(LPSTR lpDIB);
```

参数：

```
LPSTR lpbi             - 指向 DIB 对象的指针
```

返回值：

```
DWORD                  - DIB 中图像的宽度
```

说明：该函数返回 DIB 中图像的宽度。对于 Windows 3.0 DIB，返回 BITMAPINFOHEADER 中的 biWidth 值；对于其他返回 BITMAPCOREHEADER 中的 bcWidth 值。

（5）DIBHeight 函数

函数原型：

```
DWORD WINAPI DIBHeight(LPSTR lpDIB)
```

参数：

```
LPSTR lpDIB          - 指向 DIB 对象的指针
```

返回值：

```
DWORD                - DIB 中图像的高度
```

说明：该函数返回 DIB 中图像的高度。对于 Windows 3.0 DIB，返回 BITMAPINFOHEADER 中的 biHeight 值；对于其他返回 BITMAPCOREHEADER 中的 bcHeight 值。

（6）DIBNumColors 函数

函数原型：

```
WORD WINAPI DIBNumColors(LPSTR lpbi)
```

参数：

```
LPSTR lpbi           - 指向 DIB 对象的指针
```

返回值：

```
WORD                 - 返回调色板中颜色的种数
```

说明：该函数返回 DIB 中调色板的颜色的种数。对于单色位图，返回 2，对于 16 色位图，返回 16，对于 256 色位图，返回 256；对于真彩色位图（24 位），没有调色板，返回 0。

以上 6 个函数是在图像处理过程中读取/保存图像以及获取图像基本信息的 6 个最基本的函数。还有一个 NewDIB 函数是笔者自己编写的，用来建立一个新的 DIB。此函数非常有用，可以十分便利地根据所提供的要创建的位图的基本信息（高度、宽度、颜色位数）来利用内存，并自动完成位图信息头的填充工作。

（7）NewDIB 函数

其完整程序代码及注释介绍如下：

```
/*************************************************************************
*函数名称：
* NewDIB()
*
* 参数：
* width            - 将要创建 DIB 的宽
* height           - 将要创建 DIB 的高
* biBitCount       - 将要创建 DIB 的位数。比如，如果要创建 256 色 DIB，则此值为 8
*
*返回值：
*  HDIB            - 成功返回 DIB 的句柄，否则返回 NULL。
*
*说明：
*  该函数指定宽、高、颜色位数来创建一个新的 DIB，并返回其句柄
*
*************************************************************************/
HDIB WINAPI NewDIB(long width, long height,unsigned short biBitCount)
{

    //计算新建的 DIB 每行所占的字节数
    long dwindth = (width*biBitCount/8+3)/4*4;
```

```
    //新建的 DIB 调色板中表项的数目
 WORD color_num;

    //通过输入的 biBitCount 值来确定调色板的表项数目
switch(biBitCount)
{

        //如果用 1bit 来表示一个像素那么调色板中有两个表项
    case 1:
        color_num=2;
    break;

        //如果用 4bit 来表示一个像素那么调色板中有 16 个表项
    case 4:
    color_num=16;
    break;

        //如果用 8bit 来表示一个像素，那么调色板中得表项有 256 中（本程序大多采用这种形式）
    case 8:
    color_num=256;
    break;

          //其他的情况调色板中没有表项，即真彩位图
    default:
    color_num=0;
    break;
 }

    //计算位图数据所占的空间
    //dwindth *height 为像素数据所占的空间
    //40 为位图信息头占的空间
    //color_num*4 为调色板的表项所占的空间（调色板每个表项占 4 个字节）

   dwBitsSize = dwindth *height + 40 + color_num*4;

    //建立指向位图文件的指针
    LPSTR pDIB;

    //申请存储空间，并建立指向位图的句柄
    HDIB hDIB=(HDIB) ::GlobalAlloc(GMEM_MOVEABLE|GMEM_ZEROINIT, dwBitsSize);

    //如果申请空间不成功返回错误信息
 if (hDIB == 0)
 {
```

```
    return NULL;
}

    //如果申请空间成功锁定内存，并将内存的指针传给 pDIB
pDIB = (LPSTR) ::GlobalLock((HGLOBAL) hDIB);

    //建立指向位图信息头结构的指针
LPBITMAPINFO lpmf = (LPBITMAPINFO)pDIB;

    //给位图信息头内的各个参量赋值

    //指定位图信息头结构的大小为 40 字节
lpmf->bmiHeader.biSize = 40;

    //指定新建位图的宽度
lpmf->bmiHeader.biWidth = width;

    //指定新建位图的高度
    lpmf->bmiHeader.biHeight = height;

    //位平面数必须为 1
    lpmf->bmiHeader.biPlanes = 1;

    //确定新建位图表示颜色是要用到的 bit 数
    lpmf->bmiHeader.biBitCount = biBitCount;

    //是否进行压缩
lpmf->bmiHeader.biCompression = 0;

    //新建的位图中实际的位图数据所占的字节数
    lpmf->bmiHeader.biSizeImage = dwindth *height;

    //指定目标设备的水平分辨率
lpmf->bmiHeader.biXPelsPerMeter = 2925;

    //指定目标设备的垂直分辨率
    lpmf->bmiHeader.biYPelsPerMeter = 2925;

    //新建图像实际用到的颜色数 如果为 0 则用到的颜色数为 2 的 biBitCount 次
    lpmf->bmiHeader.biClrUsed = 0;

    //指定新建图像中重要的颜色数，如果为 0 则所有的颜色都重要
lpmf->bmiHeader.biClrImportant= 0;
```

```
    //如果新建的图像中含有调色板，则接下来对调色板的各种颜色分量赋初始值
    if(color_num!=0)
    {

        for(int i=0;i<color_num;i++)
        {
            lpmf->bmiColors[i].rgbRed =(BYTE)i;
            lpmf->bmiColors[i].rgbGreen =(BYTE)i;
            lpmf->bmiColors[i].rgbBlue =(BYTE)i;
        }
    }

    //解除锁定
    ::GlobalUnlock((HGLOBAL) hDIB);

    //返回新建位图的句柄
    return hDIB;

}
```

以上的 7 个函数是打开、保存、创建位图以及获取位图基本信息的常用函数，请读者熟练掌握运用。

下面要介绍的 4 个函数是显示位图、清除屏幕、以及画框的函数。其中 PaintDIB 函数是微软函数库提供的，由 DisplayDIB 函数调用，在此不再列举源代码，只是提供调用接口的讲解。其他 3 个笔者自己编写的函数将列出详细的源代码。

（8）PaintDIB 函数调用接口

函数原型：

```
BOOL WINAPI PaintDIB (HDC, LPRECT, HDIB, LPRECT, CPalette* pPal);
```

参数：

```
HDC hDC              - 输出设备 DC
LPRECT lpDCRect      - 绘制矩形区域
HDIB hDIB            - 指向 DIB 对象的指针
LPRECT lpDIBRect     - 要输出的 DIB 区域
CPalette* pPal       - 指向 DIB 对象调色板的指针
```

返回值：

```
BOOL                 - 绘制成功返回 TRUE，否则返回 FALSE。
```

说明：该函数主要用来绘制 DIB 对象。其中调用了 StretchDIBits()或者 SetDIBitsToDevice()来绘制 DIB 对象。输出的设备由由参数 hDC 指定；绘制的矩形区域由参数 lpDCRect 指定；输出 DIB 的区域由参数* lpDIBRect 指定。

（9）在屏幕上显示位图的 DisplayDIB 函数

其完整源代码及注释介绍如下：

```
/************************************ ************************************
函数名称：DisplayDIB
参数：
```

```
CDC* pDC              -指向当前设备上下文（Divice Context）的指针
HDIB hDIB             -要显示的位图的句柄
**************************************************************************/
void DisplayDIB(CDC* pDC,HDIB hDIB)
{
 //锁定位图并获取指向位图的指针
 BYTE* lpDIB=(BYTE*)::GlobalLock(hDIB);
 // 获取 DIB 宽度和高度
 int cxDIB =  ::DIBWidth((char*) lpDIB);
 int cyDIB =  ::DIBHeight((char*)lpDIB);
 //设置位图的尺寸
 CRect rcDIB,rcDest;
 rcDIB.top = rcDIB.left = 0;
 rcDIB.right = cxDIB;
 rcDIB.bottom = cyDIB;
 //设置目标客户区输出大小尺寸（在这里直接令其为位图的尺寸）
 rcDest = rcDIB;
 //清除屏幕
 ClearAll(pDC);
 //在客户区显示图像
 ::PaintDIB(pDC->m_hDC,rcDest,hDIB,rcDIB,NULL);
 //解除锁定
 ::GlobalUnlock((HGLOBAL)hDIB);
}
```

（10）清除当前屏幕客户区内容的 ClearAll 函数

其完整源代码及注释介绍如下：

```
void ClearAll(CDC *pDC)
{
 //设置清除区域
 CRect rect;
 rect.left =0;rect.top =0;rect.right =2000;rect.bottom =1000;
 //创建一白色画笔
 CPen pen;
 pen.CreatePen (PS_SOLID,1,RGB(255,255,255));
 pDC->SelectObject (&pen);
 //绘制一白色矩形以清除客户区
 pDC->Rectangle (&rect);
 //清除画笔
 ::DeleteObject (pen);
}
```

（11）画框函数 DrawFrame

其完整源代码及注释介绍如下：

```
/*************************************************************************
函数名称：DrawFrame
```

参数：

```
CDC* pDC                          -指向当前设备上下文的指针
HDIB hDIB                         -指向位图的句柄
CRectLink charRect                -一个元素为 Crect 类对象的链表
unsigned int linewidth            -指出框的宽度
COLORREF color                    -指出框的颜色
```

说明：调用此函数之前要先完成分割操作，否则无法在分割出来的字符外画框。进行分割、标准化操作之后会自动生成一个 Crect 链表。关于此链表的使用涉及到 STL(Standard Template Library)技术，在后面字符分割一节有个简要的介绍。

```
**************************************************************************/
void DrawFrame(CDC* pDC,HDIB hDIB, CRectLink charRect,unsigned int linewidth,COLORREF color)
{
 //创建画笔
 CPen pen;
 pen.CreatePen (PS_SOLID,linewidth,color);
 pDC->SelectObject (&pen);
 //创建一个NULL画刷
 ::SelectObject (*pDC,GetStockObject(NULL_BRUSH));
 CRect rect,rect2;
 //锁定位图句柄并获取其指针
 BYTE* lpDIB=(BYTE*)::GlobalLock ((HGLOBAL)hDIB);
 while(!charRect.empty())
 {
     //从表头上得到一个矩形
     rect2=rect= charRect.front();
     //从链表头上面删掉一个
     charRect.pop_front();
     //坐标转换
     //注意，这里原先的 rect 是相对于图像原点(左下角)的，
     //而在屏幕上绘图时，要转换以客户区为原点的坐标
     rect.top =::DIBHeight ((char*)lpDIB)-rect2.bottom;
     rect.bottom =::DIBHeight ((char*)lpDIB)-rect2.top ;
     pDC->Rectangle (&rect);
 }
 //解除锁定
 ::GlobalUnlock ((HGLOBAL)hDIB);
}
```

至此，图像读取/保存/创建/显示/清除屏幕/画框等11个函数已介绍完毕。

再提一下本程序自动刷新的实现。所谓自动刷新，即每当屏幕内容被遮挡或者说客户区需要重画的时候，根据 OnPaint 消息来自动的刷新客户区。这里我们使用 OnPaint 和 OnDraw 一起来实现自动刷新。代码如下：

```
void CChildView::OnPaint()
{
```

```
CPaintDC dc(this);
OnDraw(&dc);
}

//OnDraw 函数
void CChildView::OnDraw(CDC *pDC)
{
//如果 m_hDIB 不为 NULL，即表示已经加载了图像文件，那么重画客户区
if(m_hDIB!=NULL)
    DisplayDIB(pDC,m_hDIB);
}
```

下面来进入实质的图像预处理部分。

2. 图像的预处理

刚刚读入的图片如图 11-3 所示。

0123456789

图 11-3　初始读入的 256 色位图

（1）256 色位图灰度化

由于 256 色的位图的调色板内容比较复杂，使得图像处理的许多算法都没有办法展开，因此有必要对它进行灰度处理。所谓灰度图像就是图像的每一个像素的 R、G、B 分量的值是相等的。彩色图像的每个像素的 R、G、B 值是不相同的，所以显示出红绿蓝等各种颜色。灰度图像没有这些颜色差异，有的只是亮度上的不同。灰度值大的像素点比较亮（像素值最大为 255，为白色），反之比较暗（像素值最小为 0，为黑色）。图像灰度化有各种不同的算法，比较直接的一种就是给像素的 RGB 值各自一个加权系数，然后求和；同时还要对调色板表项进行相应的处理。

注意：最后得到的结果一定要归一到 0 ~ 255 之内，因为这是每个字节表示图像数据的极限。

图像的灰度处理由 Convert256toGray 函数完成。

```
/**********************************************************************
* 函数名称：
*      Convert256toGray()
*
* 参数：
*      HDIB hDIB      —图像的句柄
*
* 返回值：
*          无
*
* 功能：
```

```
*      将 256 色位图转化为灰度图
*
*****************************************************************/
void Convert256toGray(HDIB hDIB)
{
 LPSTR   lpDIB;

 // 由 DIB 句柄得到 DIB 指针并锁定 DIB
 lpDIB = (LPSTR) ::GlobalLock((HGLOBAL)hDIB);

 // 指向 DIB 像素数据区的指针
 LPSTR   lpDIBBits;

 // 指向 DIB 像素的指针
 BYTE *  lpSrc;

 // 图像宽度
 LONG    lWidth;
 // 图像高度
 LONG    lHeight;

 // 图像每行的字节数
 LONG    lLineBytes;

 // 指向 BITMAPINFO 结构的指针（Win3.0）
 LPBITMAPINFO lpbmi;

 // 指向 BITMAPCOREINFO 结构的指针
 LPBITMAPCOREINFO lpbmc;

 // 获取指向 BITMAPINFO 结构的指针（Win3.0）
 lpbmi = (LPBITMAPINFO)lpDIB;

 // 获取指向 BITMAPCOREINFO 结构的指针
 lpbmc = (LPBITMAPCOREINFO)lpDIB;

 // 灰度映射表
 BYTE bMap[256];

 // 计算灰度映射表（保存各个颜色的灰度值），并更新 DIB 调色板
 int i,j;
 for (i = 0; i < 256; i ++)
 {
     // 计算该颜色对应的灰度值
```

```
    bMap[i] = (BYTE)(0.299 * lpbmi->bmiColors[i].rgbRed +
                     0.587 * lpbmi->bmiColors[i].rgbGreen +
                     0.114 * lpbmi->bmiColors[i].rgbBlue + 0.5);
    // 更新 DIB 调色板红色分量
    lpbmi->bmiColors[i].rgbRed = i;

    // 更新 DIB 调色板绿色分量
    lpbmi->bmiColors[i].rgbGreen = i;

    // 更新 DIB 调色板蓝色分量
    lpbmi->bmiColors[i].rgbBlue = i;

    // 更新 DIB 调色板保留位
    lpbmi->bmiColors[i].rgbReserved = 0;

}
// 找到 DIB 图像像素起始位置
lpDIBBits = ::FindDIBBits(lpDIB);

// 获取图像宽度
lWidth = ::DIBWidth(lpDIB);

// 获取图像高度
lHeight = ::DIBHeight(lpDIB);

// 计算图像每行的字节数
lLineBytes = WIDTHBYTES(lWidth * 8);

// 更换每个像素的颜色索引（即按照灰度映射表换成灰度值）

//逐行扫描
for(i = 0; i < lHeight; i++)
{

        //逐列扫描
    for(j = 0; j < lWidth; j++)
    {
        // 指向 DIB 第 i 行，第 j 个像素的指针
        lpSrc = (unsigned char*)lpDIBBits + lLineBytes * (lHeight - 1 - i)+ j;

        // 变换
        *lpSrc = bMap[*lpSrc];
```

```
    }
  }

  //解除锁定
  ::GlobalUnlock ((HGLOBAL)hDIB);
}
```

下面来编写由 256 色位图转化为灰度位图的菜单处理事件的代码：

```
//图像预处理第 1 步：将 256 色图像转化为灰度图像
void CChildView::OnIMGPRC256ToGray()
{
   //调用灰度转化函数
 Convert256toGray(m_hDIB);

 //在屏幕上显示位图
 CDC* pDC=GetDC();
 DisplayDIB(pDC,m_hDIB);
}
```

经过灰度处理后的 256 色位图如图 11-4 所示。

0123456789

图 11-4　经过灰度处理的文件

（2）灰度图像二值化

在进行了灰度化处理之后，图像中的每个像素只有一个值，那就是像素的灰度值。它的大小决定了像素的亮暗程度。为了更加便利的开展下面的图像处理操作，还需要对已经得到的灰度图像做一个二值化处理。图像的二值化就是把图像中的像素根据一定的标准分化成两种颜色。在系统中是根据像素的灰度值处理成黑白两种颜色。和灰度化相似的，图像的二值化也有很多成熟的算法。它可以采用自适应阀值法，也可以采用给定阀值法。系统中采用的是给定阀值的方法。因为考虑到所要进行处理的图像大多是从印刷出版物上扫描得来的底色大多为白色所以我们将这个阈值固定为 220，读者也可以根据实际的情况来自己进行阈值的设定。

图像二值化的函数由 ConvertGrayToWhiteBlack 实现，源代码如下：

```
/***********************************************************************
*
* 函数名称 ConvertGrayToWhiteBlack  ()
*
* 参数 : HDIB hDIB      —原图的句柄
*
* 返回值：无
*
* 功能：ConvertGrayToWhiteBlack 函数采用硬阈值的方法，实现将图像二值化的功能。
```

```
*
* 说明:
要求待处理的图片为256色
*************************************************************************/
void ConvertGrayToWhiteBlack(HDIB hDIB)
{
 // 指向DIB的指针
 LPSTR   lpDIB;

 // 由DIB句柄得到DIB指针并锁定DIB
 lpDIB = (LPSTR) ::GlobalLock((HGLOBAL)hDIB);

 // 指向DIB像素数据区的指针
 LPSTR   lpDIBBits;

 // 指向DIB像素的指针
 BYTE *  lpSrc;

 // 图像宽度
 LONG    lWidth;

 // 图像高度
 LONG    lHeight;
 // 图像每行的字节数
 LONG    lLineBytes;

 // 找到DIB图像像素起始位置
 lpDIBBits = ::FindDIBBits(lpDIB);

 // 获取图像宽度
 lWidth = ::DIBWidth(lpDIB);

 // 获取图像高度
 lHeight = ::DIBHeight(lpDIB);

 // 计算图像每行的字节数
 lLineBytes = WIDTHBYTES(lWidth * 8);

 // 更换每个像素的颜色索引(即按照灰度映射表换成灰度值)
 int i,j;

    //逐行扫描
 for(i = 0; i < lHeight; i++)
    {
```

```
        //逐列扫描
        for(j = 0; j < lWidth; j++)
        {

            // 指向 DIB 第 i 行，第 j 个像素的指针
            lpSrc = (unsigned char*)lpDIBBits + lLineBytes * i + j;

            // 二值化处理
            //大于 220，设置为 255，即白点
            if(*lpSrc>220) *lpSrc=255;

            //否则设置为 0，即黑点
            else *lpSrc=0;

        }
    }

    //解除锁定
    ::GlobalUnlock((HGLOBAL)hDIB);
}
```

下面编写对灰度图像进行二值化处理的菜单处理时间的代码：

```
//图像预处理第 2 步：将灰度图二值化
void CChildView::OnIMGPRCGrayToWhiteBlack()
{
    //调用灰度二值化函数
    ConvertGrayToWhiteBlack(m_hDIB);

 //在屏幕上显示位图
 CDC* pDC=GetDC();
 DisplayDIB(pDC,m_hDIB);
}
```

执行结果如图 11-5 所示。

图 11-5　二值化后的图片

（3）图像的梯度锐化

由于需要处理的图像大多由印刷出版物上扫描而来所以在很多的情况下字体模糊，对识别造成了一定的困难，所以有时要对图像进行锐化处理使模糊的图像变得清晰，同时可以对噪声起到一定的去除作用。图像锐化的方法有很多，有一种是微分法，有一种是高通滤波法，我们

在这里所采用的梯度锐化的方法就属于微分法的一种。在这里采用 Roberts 梯度算子对图像进行锐化。

定义如下：设原始图像上的点为 $f(x,y)$，定义 $f(x,y)$ 在 (x,y) 处的梯度矢量为：

$$G[f(i,j)]=|f(i,i)-f(i+1,j)|+|f(i,j)-f(i,j+1)| \tag{11-1}$$

设一个判定阈值为 Δ，变化后的图像 $g(x,y)$ 定义为：

$$g(x,y)=\begin{cases}G[f(x,y)] & (G[f(x,y)]\geqslant\Delta)\\ f(x,y) & (G[f(x,y)]\leqslant\Delta)\end{cases} \tag{11-2}$$

通过公式可以看出梯度锐化可以让模糊的边缘变得清楚 同时选择合适的阈值还可以减弱和消除一些细小的噪声。本程序给出了梯度锐化的完整代码，读者也可以根据实际需求、读入图片的质量来选择决定是否使用梯度锐化。事实证明，梯度锐化具备一定的去噪声能力，但同时会对字符的边缘有所损伤。所以笔者建议在图片中字符较为细小的时候不要使用梯度锐化。

图像的梯度锐化可以通过函数 GradientSharp 来实现：

```
/************************************************
*
* 函数名称：
*   radientSharp ()
*
*参数 ：
*  HDIB hDIB     -待处理图像的句柄
*
* 返回值：
*        无
*
*功能：
*     现图像的梯度锐化

  *说明：
  *     只能对 2 值图像进行处理，如果图像本身边缘较细，可能造成信息的损失
*************************************************************************/
void GradientSharp(HDIB hDIB)
{
 // 指向 DIB 的指针
 LPSTR lpDIB=(LPSTR) ::GlobalLock((HGLOBAL)hDIB);

 // 指向 DIB 像素指针
 LPSTR    lpDIBBits;

 // 找到 DIB 图像像素起始位置
 lpDIBBits = ::FindDIBBits(lpDIB);
```

```
//获取图像的宽度
LONG lWidth=::DIBWidth ((char*)lpDIB);

//获取图像的长度
LONG lHeight=::DIBHeight ((char*)lpDIB);

// 阈值
BYTE    bThre = 2;

// 调用 GradSharp()函数进行梯度板锐化

// 指向源图像的指针
unsigned char*  lpSrc;
unsigned char*  lpSrc1;
unsigned char*  lpSrc2;

// 循环变量
LONG    i;
LONG    j;

// 图像每行的字节数
LONG    lLineBytes;

// 中间变量
BYTE    bTemp;

// 计算图像每行的字节数
lLineBytes = WIDTHBYTES(lWidth * 8);

// 每行
for(i = 0; i < lHeight; i++)
{

    // 每列
    for(j = 0; j < lWidth; j++)
    {

        // 指向 DIB 第 i 行，第 j 个像素的指针
        lpSrc  = (unsigned char*)lpDIBBits + lLineBytes * (lHeight - 1 - i)+j;

        // 指向 DIB 第 i+1 行，第 j 个像素的指针
        lpSrc1 = (unsigned char*)lpDIBBits + lLineBytes * (lHeight - 2 - i)+j;
```

```
        // 指向 DIB 第 i 行，第 j+1 个像素的指针
    lpSrc2 = (unsigned char*)lpDIBBits + lLineBytes * (lHeight - 1 - i)+j+ 1;

        //计算梯度值
        bTemp = abs((*lpSrc)-(*lpSrc1)) + abs((*lpSrc)-(*lpSrc2));

        // 判断是否小于阈值
        if (bTemp < 255)
        {
            // 判断是否大于阈值，对于小于情况，灰度值不变。
            if (bTemp >= bThre)
            {
                // 直接赋值为 bTemp
                lpSrc = bTemp;
            }
        }
        else
        {
            // 直接赋值为 255
            *lpSrc = 255;
        }
    }
}

//最后还要处理一下图像中最下面那行
for(j = 0; j < lWidth; j++)
{
    //指向最下边一行，第 j 个像素的指针
    lpSrc  = (unsigned char*)lpDIBBits + lLineBytes * 0 + j;

    //将此位置的像素设置为 255，即白点
    *lpSrc=255;
}

//解除锁定
::GlobalUnlock ((HGLOBAL)hDIB);
}
```

下面我们来编写对图像进行梯度锐化菜单处理事件的响应代码：

```
//图像预处理第 3 步：梯度锐化
void CChildView::OnImgprcSharp()
{
    //调用梯度锐化函数
GradientSharp(m_hDIB);
```

```
//在屏幕上显示位图
CDC* pDC=GetDC();
DisplayDIB(pDC,m_hDIB);
}
```

经过梯度锐化处理后的图片如图 11-6 所示。

0123456789

图 11-6　梯度锐化后的图像

从处理的结果也可以看出图像的边缘变得清晰而且少了很多细小的杂点，但是梯度锐化有其自身的缺陷。当处理的图像边缘很细的时候可能造成边缘信息的损失（如图 11-7 所示），所以务必提醒读者注意，要根据实际情况酌情处理。

原始图像 0 1 2 3 4 5 6 7 8 9

梯度锐化以后 0 1 2 3 4 5 6 7 8 9

图 11-7　对边缘较细的图像进行梯队锐化的结果

（4）去除离散的杂点噪声

图像可能在扫描或者传输过程中夹带了噪声，去噪声是图像处理中常用的手法。通常去噪声用滤波的方法，比如中值滤波、均值滤波。但是那样的算法不适合用在处理字符这样目标狭长的图像中，因为在滤波的过程中很有可能会去掉字符本身的像素。本系统采用的是去除杂点的方法来进行去噪声处理的。具体算法如下：扫描整个图像，当发现一个黑色点的时候，就考察和该黑色点间接或者直接相连接的黑色点的个数有多少，如果大于一定的值，那就说明该点不是离散点，否则就是离散点，把它去掉。在考察相连的黑色点的时候用的是递归的方法。

去杂点的功能由 RemoveScatterNoise、DeleteScaterJudge 两个函数完成，下面分别介绍两个函数。

函数 RemoveScatterNoise 代码如下：

```
/****************************************************************
*
*  函数名称:
*        RemoveScatterNoise()
*
*  参数:
*      HDIB    hDIB      —原图像的句柄
*
*  返回值:
*        无
*
*  功能:
```

```
*      通过对连续点长度的统计来去除离散杂点
*
* 说明:
*      只能对 2 值图像进行处理
*************************************************************/
void RemoveScatterNoise(HDIB hDIB)
{

 // 指向 DIB 的指针
 LPSTR lpDIB=(LPSTR) ::GlobalLock((HGLOBAL)hDIB);

 // 指向 DIB 像素指针
 LPSTR    lpDIBBits;

 // 找到 DIB 图像像素数据区的起始位置
 lpDIBBits = ::FindDIBBits(lpDIB);

 //获得图像的长度
 LONG lWidth=::DIBWidth ((char*)lpDIB);

 //获得图像的高度
 LONG lHeight=::DIBHeight ((char*)lpDIB);

 //设置判定噪声的长度阈值为 15
 //即如果与考察点相连接的黑点的数目小于 15 则认为考察点是噪声点
 int length=15;

 // 循环变量
 m_lianXuShu=0;
 LONG    i;
 LONG    j;
 LONG    k;

 // 图像每行的字节数
 LONG    lLineBytes;

 // 计算图像每行的字节数
 lLineBytes = WIDTHBYTES(lWidth * 8);


 LPSTR lpSrc;

 //开辟一块用来存放标志的内存数组
 LPBYTE lplab = new BYTE[lHeight * lWidth];
```

```
//开辟一块用来保存离散判定结果的内存数组
bool *lpTemp = new bool[lHeight * lWidth];

//初始化标志数组
for (i=0;i<lHeight*lWidth;i++)
{

    //将所有的标志位设置为非
    lplab[i] = false;

}

//用来存放离散点的坐标的数组
CPoint lab[21];

//为循环变量赋初始值
k=0;

//扫描整个图像

//逐行扫描
for(i =0;i<lHeight;i++)
{

    //逐行扫描
    for(j=0;j<lWidth;j++)
    {
        //先把标志位置false
        for(k=0;k<m_lianXuShu;k++)
            lplab[lab[k].y * lWidth + lab[k].x] = false;

        //连续数置0
        m_lianXuShu =0;

        //进行离散性判断
 lpTemp[i*lWidth+j] = DeleteScaterJudge(lpDIBBits,(WORD)lLineBytes, lplab,
lWidth,lHeight,j,i,lab,length);

    }
}

//扫描整个图像，把离散点填充成白色
```

```
//逐行扫描
for(i = 0;i<lHeight;i++)
{

    //逐列扫描
    for(j=0;j<lWidth;j++)
    {
        //查看标志位,如果为非则将此点设为白点
        if(lpTemp[i*lWidth+j] == false)
        {
            //指向第 i 行第 j 个像素的指针
            lpSrc=(char*)lpDIBBits + lLineBytes * i + j;

            //将此像素设为白点
            *lpSrc=BYTE(255);
        }
    }
}

//解除锁定
::GlobalUnlock ((HGLOBAL)hDIB);
}
```

函数 DeleteScaterJudge 的代码如下：

```
/*************************************************************************
*
*  函数名称
*       DeleteScaterJudge()
*
*  参数:
*     LPSTR   lpDIBBits       一指向像素起始位置的指针
*     WORD    lLineBytes      一图像每行的字节数
*     LPBYTE  lplab           一标志位数组
*     int     lWidth          一图像的宽度
*     int     lHeight         一图像的高度
*     int     x               一当前点的横坐标
*     int     y               一当前点的纵坐标
*     CPoint  lab[]           一存放已考察过的连续点坐标
*     int     lianXuShu       一离散点的判定长度
*
*  返回值:
*     Bool                    一是离散点返回 false 不是离散点返回 true
*
*  功能:
*     利用递归算法统计连续点的个数，通过阈值来判定是否为离散点
```

```
 *
 * 说明:
 *     只能对 2 值图像进行处理
 *************************************************************************/
bool DeleteScaterJudge(LPSTR lpDIBBits, WORD lLineBytes, LPBYTE lplab, int
lWidth, int lHeight, int x, int y, CPoint lab[], int lianXuShu)
{

 //如果连续长度满足要求，说明不是离散点，返回
 if(m_lianXuShu>=lianXuShu)
     return TRUE;

 //长度加一
 m_lianXuShu++;

 //设定访问标志
 lplab[lWidth * y +x] = true;

 //保存访问点坐标
 lab[m_lianXuShu-1].x = x;
 lab[m_lianXuShu-1].y = y;

 //像素的灰度值
 int gray;

 //指向像素的指针
 LPSTR lpSrc;

 //长度判定
 //如果连续长度满足要求，说明不是离散点，返回
 if(m_lianXuShu>=lianXuShu)
     return TRUE;

 //下面进入递归
 else
 {
     //考察上下左右以及左上、右上、左下、右下八个方向
     //如果是黑色点，则调用函数自身进行递归

     //考察下面点

     lpSrc=(char*)lpDIBBits + lLineBytes * (y-1) + x;

     //传递灰度值
```

```
        gray=*lpSrc;

        //如果点在图像内、颜色为黑色并且没有被访问过
        if(y-1 >=0 && gray == 0 && lplab[(y-1)*lWidth+x] == false)

            //进行递归处理
    DeleteScaterJudge(lpDIBBits,lLineBytes,lplab,lWidth,lHeight,x,y-1,lab,lian
XuShu);

            //判断长度

            //如果连续长度满足要求，说明不是离散点，返回
            if(m_lianXuShu>=lianXuShu)
                return TRUE;

            //左下点

            lpSrc=(char*)lpDIBBits + lLineBytes * (y-1) + x-1;

            //传递灰度值
            gray=*lpSrc;

            //如果点在图像内、颜色为黑色并且没有被访问过
            if(y-1 >=0 &&  x-1 >=0 && gray== 0 && lplab[(y-1)*lWidth+x-1] == false)

                //进行递归处理
    DeleteScaterJudge(lpDIBBits,lLineBytes,lplab,lWidth,lHeight,x-1,y-1,lab,li
anXuShu);

                //判断长度

                //如果连续长度满足要求，说明不是离散点，返回
                if(m_lianXuShu>=lianXuShu)
                    return TRUE;

                //左边

                lpSrc=(char*)lpDIBBits + lLineBytes * y + x-1;

                //传递灰度值
                gray=*lpSrc;

                //如果点在图像内、颜色为黑色并且没有被访问过
                if(x-1 >=0 &&  gray== 0 && lplab[y*lWidth+x-1] == false)
```

```
                    //进行递归处理
    DeleteScaterJudge(lpDIBBits,lLineBytes,lplab,lWidth,lHeight,x-1,y,lab,lian
XuShu);

                    //判断长度

                    //如果连续长度满足要求，说明不是离散点，返回
                    if(m_lianXuShu>=lianXuShu)
                        return TRUE;

                    //左上

                    lpSrc=(char*)lpDIBBits + lLineBytes * (y+1) + x-1;

                    //传递灰度值
                    gray=*lpSrc;

                    //如果点在图像内、颜色为黑色并且没有被访问过
                    if(y+1 <lHeight && x-1 >= 0 && gray == 0 && lplab[(y+1)*lWidth+x
-1] == false)

                        //进行递归处理
    DeleteScaterJudge(lpDIBBits,lLineBytes,lplab,lWidth,lHeight,x-1,y+1,lab,li
anXuShu);

                        //判断长度

                        //如果连续长度满足要求，说明不是离散点，返回
                        if(m_lianXuShu>=lianXuShu)
                            return TRUE;

                        //上面

                        lpSrc=(char*)lpDIBBits + lLineBytes * (y+1) + x;

                        //传递灰度值
                        gray=*lpSrc;

                        //如果点在图像内、颜色为黑色并且没有被访问过
                    if(y+1 < lHeight && gray == 0 && lplab[(y+1)*lWidth+x] == false)

                            //进行递归处理
    DeleteScaterJudge(lpDIBBits,lLineBytes,lplab,lWidth,lHeight,x,y+1,lab,lian
```

```
XuShu);

                        //判断长度

                        //如果连续长度满足要求，说明不是离散点，返回
                        if(m_lianXuShu>=lianXuShu)
                            return TRUE;

                        //右上

                        lpSrc=(char*)lpDIBBits + lLineBytes * (y+1) + x+1;

                        //传递灰度值
                        gray=*lpSrc;

                        //如果点在图像内、颜色为黑色并且没有被访问过
                        if(y+1 <lHeight && x+1 <lWidth &&  gray == 0 &&
lplab[(y+1)*lWidth+x+1] == false)

                            //进行递归处理

    DeleteScaterJudge(lpDIBBits,lLineBytes,lplab,lWidth,lHeight,x+1,y+1,lab,li
anXuShu);

                        //判断长度

                        //如果连续长度满足要求，说明不是离散点，返回
                        if(m_lianXuShu>=lianXuShu)
                            return TRUE;

                        //右边
                        lpSrc=(char*)lpDIBBits + lLineBytes * y + x+1;

                        //传递灰度值
                        gray=*lpSrc;

                        //如果点在图像内、颜色为黑色并且没有被访问过
                    if(x+1 <lWidth && gray==0 && lplab[y*lWidth+x+1] == false)

                            //进行递归处理
    DeleteScaterJudge(lpDIBBits,lLineBytes,lplab,lWidth,lHeight,x+1,y,lab,lian
XuShu);

                            //判断长度
```

```
                                        //如果连续长度满足要求，说明不是离散点，返回
                                        if(m_lianXuShu>=lianXuShu)
                                            return TRUE;

                                        //右下

                                        lpSrc=(char*)lpDIBBits + lLineBytes * (y-1) + x+1;

                                        //传递灰度值
                                        gray=*lpSrc;

                                        //如果点在图像内、颜色为黑色并且没有被访问过
                                        if(y-1  >=0  &&  x+1  <lWidth  &&  gray  ==  0  &&
lplab[(y-1)*lWidth+x+1] == false)

                                            //进行递归处理
    DeleteScaterJudge(lpDIBBits,lLineBytes,lplab,lWidth,lHeight,x+1,y-1,lab,li
anXuShu);

                                            //判断长度

                                            //如果连续长度满足要求，说明不是离散点，返回
                                            if(m_lianXuShu>=lianXuShu)
                                                return TRUE;
    }

    //如果递归结束，返回 false，说明是离散点
    return FALSE;

    }
```

下面编写去除图像中离散杂点噪声的菜单处理事件的代码：

```
    //图像预处理第 4 步：去离散杂点噪声
    void CChildView::OnImgprcRemoveNoise()
    {
     //调用去离散杂点噪声的函数
     RemoveScatterNoise(m_hDIB);

     //在屏幕上显示位图
     CDC* pDC=GetDC();
     DisplayDIB(pDC,m_hDIB);
    }
```

执行后的结果如图 11-8 所示。

0123456789

图 11-8 去除离散杂点噪声后的图像

（5）整体倾斜度调整

因为读进来的图像可能存在倾斜，所以必须对它进行调整，使得字符都处于同一水平位置，那样既有利于字符的分割也可以提高字符识别的准确率。调整的算法主要是根据图像上左右两边的黑色像素的平均高度来的。一般来说，众多的字符组成的图像它的左右两边的字符像素的高度应该是处于水平位置附近的，如果两边字符像素的平均位置有比较大的起落，那就说明图像存在倾斜，需要进行调整。具体来说，首先要分别计算图像左半边和右半边的像素的平均高度，然后求斜率，根据斜率重新组织图像，里面包含了一个从新图像到旧图像的像素的映射。如果新图像中的像素映射到旧图像中时超出了旧图像的范围，则把新图像中的该像素置白色。

图像的调整由 SlopeAdjust 函数完成。

SlopeAdjust 函数代码如下：

```
/****************************************************************

 * 函数名称：
 *         SlopeAdjust()
 *
 * 参数：
 *     HDIB   hDIB        一原图像的句柄
 *
 * 返回值：
 *         无
 *
 * 功能：
 *     通过对图像左右半边平均高度的统计来进行倾斜的调整
 *
 * 说明：
 *      只能对 2 值图像进行处理
 *
****************************************************************/
void SlopeAdjust(HDIB hDIB)
{
 // 指向 DIB 的指针
 LPSTR lpDIB=(LPSTR) ::GlobalLock((HGLOBAL)hDIB);

 // 指向 DIB 像素指针
 LPSTR   lpDIBBits;
```

```
// 找到 DIB 图像像素起始位置
lpDIBBits = ::FindDIBBits(lpDIB);

// 指向源图像的指针
unsigned char*  lpSrc;

// 循环变量
LONG    i;
LONG    j;

// 图像每行的字节数
LONG    lLineBytes;

//图像的长度
LONG    lWidth;

//图像的宽度
LONG    lHeight;

//获取图像的长度
lWidth=::DIBWidth ((char*)lpDIB);

//获取图像的宽度
lHeight=::DIBHeight ((char*)lpDIB);

// 计算图像每行的字节数
lLineBytes = WIDTHBYTES(lWidth * 8);

//图像左半边的平均高度
double leftaver=0.0;

    //图像右半边的平均高度
    double rightaver=0.0;

//图像的倾斜度
double slope;

//统计循环变量
LONG counts=0;

//扫描左半边的图像，求黑色像素的平均高度

//行
for (i=0;i<lHeight;i++)
```

```
{

    //列
    for (j=0;j<lWidth/2;j++)
    {

        //指向第 i 行第 j 个像素的指针
        lpSrc=(unsigned char*)lpDIBBits + lLineBytes *  i + j;

        //如果为黑点
        if (*lpSrc == 0)
        {

            //对其高度进行统计叠加
            counts +=lWidth/2 -j;
            leftaver += i*(lWidth/2 -j);

        }

    }
}

//计算平均高度
leftaver /= counts;

//将统计循环变量重新赋值
counts =0;

//扫描右半边的图像，求黑色像素的平均高度

//行
for (i =0;i<lHeight;i++)
{

    //列
    for (j=lWidth/2;j<lWidth;j++)
    {
        //指向第 i 行第 j 个像素的指针
        lpSrc=(unsigned char*)lpDIBBits + lLineBytes *  i + j;

        //如果为黑点
        if (*lpSrc == 0)
        {
```

```
            //进行统计叠加
            counts +=lWidth -j;
            rightaver += i*(lWidth -j);
        }
    }
}

//计算右半边的平均高度
rightaver /= counts;

//计算斜率
slope = (leftaver - rightaver) / (lWidth/2);

//指向新的图像像素起始位置的指针
LPSTR lpNewDIBBits;

//指向新图像的指针
LPSTR lpDst;

//新图像的句柄
HLOCAL nNewDIBBits=LocalAlloc(LHND,lLineBytes*lHeight);

//锁定内存
lpNewDIBBits=(char*)LocalLock(nNewDIBBits);

//指向新图像像素的指针
lpDst=(char*)lpNewDIBBits;

//为新图像赋初始值
memset(lpDst,(BYTE)255,lLineBytes*lHeight);

//像素点的灰度值
int gray;

//位置映射值
int i_src;

//根据斜率，把当前新图像的点映射到源图像的点

//行
for (i=0;i<lHeight;i++)
{
    //列
    for (j=0;j<lWidth;j++)
```

```
    {
        //计算映射位置
        i_src=int(i - (j-lWidth/2)*slope);

        //如果点在图像外，像素置白色
        if (i_src <0 || i_src >=lHeight )
            gray = 255;

        else
        {
            //否则到源图像中找点，取得像素值

            //指向第 i_src 行第 j 个像素的指针
            lpSrc=(unsigned char *)lpDIBBits + lLineBytes * i_src + j;
            gray = *lpSrc;
        }

        //把新图像的点用得到的像素值填充

        //指向第 i 行第 j 个像素的指针
        lpDst = (char *)lpNewDIBBits + lLineBytes * i + j;
        *lpDst=gray;
    }
}

// 将新的图像的内容拷贝到旧的图像中
memcpy(lpDIBBits,lpNewDIBBits,lLineBytes*lHeight);

// 解除锁定
::GlobalUnlock ((HGLOBAL)hDIB);
}
```

下面编写倾斜度调整菜单处理事件的代码：

```
//图像预处理第 5 步：倾斜度调整
void CChildView::OnImgprcAdjustSlope()
{
    //调用倾斜度调整函数
 SlopeAdjust(m_hDIB);

 //在屏幕上显示位图
 CDC* pDC=GetDC();
 DisplayDIB(pDC,m_hDIB);
}
```

经过倾斜度调整后的图像如图 11-9 所示。

0123456789

图 11-9　经过倾斜度调整后的图像

（6）字符分割

系统在读进来的图像中一般会含有多个数字，识别的时候只能根据每个字符的特征来进行判断，所以还要进行字符分割的工作。这一步工作就是把图像中的字符独立的分割出来。

具体的算法如下：

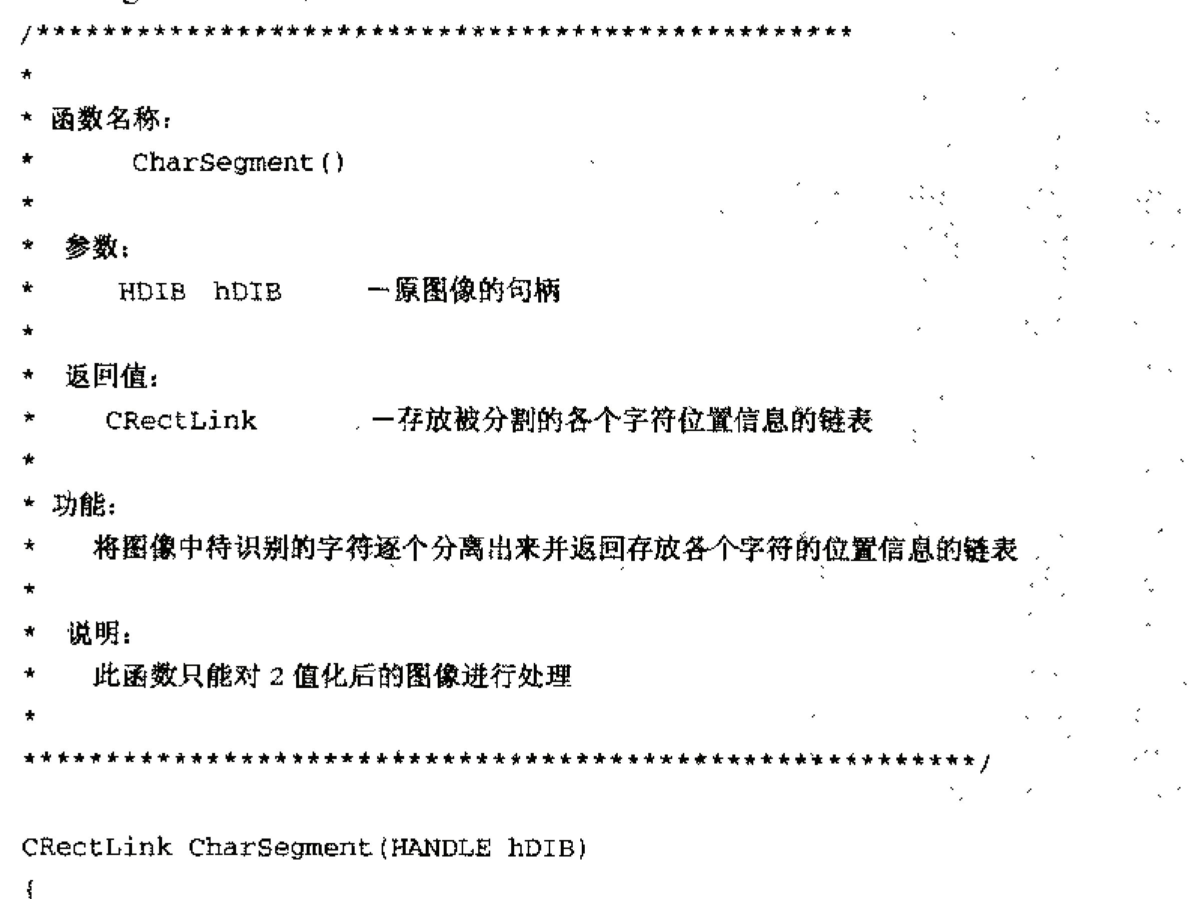
第 1 步，先自下向上对图像进行逐行扫描直至遇到第一个黑色的像素点。记录下来。然后再由上向下对图像进行逐行扫描直至找到第一个黑色像素，这样就找到图像大致的高度范围。

第 2 步，在这个高度范围之内在自左向右逐列进行扫描，遇到第一个黑色像素时认为是字符分割的起始位置，然后继续扫描，直至遇到有一列中没有黑色像素，则认为这个字符分割结束，然后继续扫描，按照上述的方法一直扫描直至图像的最右端。这样就得到了每个字符的比较精确宽度范围。

第 3 步，在已知的每个字符比较精确的宽度范围内，按照第一步的方法，分别进行自上而下和自下而上的逐行扫描来获取每个字符精确的高度范围。

字符的分割用 CharSegment 函数来实现。

CharSegment 函数代码如下：



```
/*************************************************
*
* 函数名称：
*       CharSegment()
*
*  参数：
*      HDIB  hDIB       —原图像的句柄
*
*  返回值：
*     CRectLink         —存放被分割的各个字符位置信息的链表
*
* 功能：
*    将图像中待识别的字符逐个分离出来并返回存放各个字符的位置信息的链表
*
*  说明：
*    此函数只能对 2 值化后的图像进行处理
*
*********************************************************/

CRectLink CharSegment(HANDLE hDIB)
{
```

```
//清空用来保存每个字符区域的链表
CRectLink charRect1,charRect2;
charRect1.clear();
charRect2.clear();

// 指向 DIB 的指针
LPSTR lpDIB=(LPSTR) ::GlobalLock((HGLOBAL)hDIB);

// 指向 DIB 像素指针
LPSTR    lpDIBBits;

// 找到 DIB 图像像素起始位置
lpDIBBits = ::FindDIBBits(lpDIB);

//指向像素的指针
BYTE* lpSrc;

//图像的长度和宽度
int height,width;

//获取图像的宽度
width=(int)::DIBWidth(lpDIB);

//获取图像的长度
height=(int)::DIBHeight(lpDIB);

//计算图像每行的字节数
LONG    lLineBytes = WIDTHBYTES(width * 8);

//定义上下边界两个变量
int top,bottom;

//像素的灰度值
int gray;

//设置循环变量
int i,j;

//用来统计图像中字符个数的计数器
digicount=0;


//从上往下扫描，找到上边界
```

```
//行
for (i=0;i<height;i++)
{
    //列
    for (j=0;j<width;j++)
    {
        // 指向图像第 i 行，第 j 个像素的指针
        lpSrc = (unsigned char*)lpDIBBits + lLineBytes * i + j;

        //获得该点的灰度值
        gray = *(lpSrc);

        //看是否为黑点
        if (gray == 0)
        {
            //若为黑点，把此点作为字符大致的最高点
            top = i;

            //对 i 强行赋值以中断循环
            i=height;

            //跳出循环
            break;
        }

        //如果该点不是黑点，继续循环

    }
}


//从下往上扫描，找下边界

//行
for (i = height-1;i>=0;i--)
{

    //列
    for (j=0;j<width;j++)
    {
        // 指向图像第 i 行，第 j 个像素的指针
        lpSrc = (unsigned char*)lpDIBBits + lLineBytes * i + j;

        //获取该点的灰度值
```

```
            gray = *(lpSrc);

            //判断是否为黑点
            if (gray == 0)
            {
                //若为黑点，把此点作为字符大致的最低点
                bottom = i;

                //对 i 强行赋值以中断循环
                i=-1;

                //跳出循环
                break;
            }

            //如果该点不是黑点，继续循环
        }

    }

    //lab 用作是否进入一个字符分割的标志
    bool lab = false;

    //表明扫描一列中是否发现黑色点
    bool black = false;

    //存放位置信息的结构体
    CRect rect;

    //计数器置零
    digicount=0;

    //行
    for (i=0;i<width;i++)
    {
        //开始扫描一列
        black=false;

        for (j=0;j<height;j++)
        {
            // 指向图像第 i 行，第 j 个像素的指针
            lpSrc = (unsigned char*)lpDIBBits + lLineBytes * j + i;

            //获取该点的灰度值
```

```
            gray = *(lpSrc);

            //判断是否为黑点
            if (gray == 0)
            {
                //如果发现黑点，设置标志位
                black=true;

                //如果还没有进入一个字符的分割
                if(lab==false)
                {
                    //设置左侧边界
                    rect.left = i;

                    //字符分割开始
                    lab = true;
                }

                //如果字符分割已经开始了
                else

                    //跳出循环
                    break;
            }
        }

        //如果已经扫到了最右边那列，说明整副图像扫描完毕。退出
        if(i==(width-1))

            //退出整个循环
            break;

        //如果到此 black 仍为 false，说明扫描了一列，都没有发现黑点。表明当前字符分割结束
        if(lab==true&&black==false)
        {
            //将位置信息存入结构体中

            //设置右边界
            rect.right =i;

            //设置上边界
            rect.top =top;

            //设置下边界
```

```
            rect.bottom =bottom;

            //将框外括一个像素，以免压到字符
            rect.InflateRect (1,1);

            //将这个结构体插入存放位置信息的链表 1 的后面
            charRect1.push_back (rect);

            //设置标志位，开始下一次的字符分割
            lab=false;

            //字符个数统计计数器加 1
            digicount++;

        }

        //进入下一列的扫描

    }

    //再将矩形轮廓矩形的 top 和 bottom 精确化

    //将链表 1 赋值给链表 2
    charRect2=charRect1;

    //将链表 2 的内容清空
    charRect2.clear ();

    //建立一个新的存放位置信息的结构体
    CRect rectnew;

    //对于链表 1 从头至尾逐个进行扫描
    while(!charRect1.empty())
    {
        //从链表 1 头上得到一个矩形
        rect= charRect1.front();

        //从链表 1 头上面删掉一个
        charRect1.pop_front();

        //计算更加精确的矩形区域

        //获得精确的左边界
        rectnew.left =rect.left-1 ;
```

```
//获得精确的右边界
rectnew.right =rect.right+1 ;

//通过获得的精确左右边界对上下边境重新进行精确定位

// 由下而上扫描计算上边界

//行
for(i=rect.top ;i<rect.bottom ;i++)
{
    //列
    for(j=rect.left ;j<rect.right ;j++)
    {
        // 指向图像第 i 行，第 j 个像素的指针
        lpSrc = (unsigned char*)lpDIBBits + lLineBytes * i + j;

        //如果这个像素是黑点
        if (*lpSrc == 0)
        {
            //设置上边界
            rectnew.top = i-1;

            //对 i 进行强制定义以跳出循环
            i=rect.bottom  ;

            //跳出循环
            break;
        }
    }
}

//由下而上扫描计算下边界

//行
for(i=rect.bottom-1 ;i>=rect.top  ;i--)
{
    //列
    for(j=rect.left ;j<rect.right ;j++)
    {
        // 指向图像第 i 行，第 j 个像素的指针
        lpSrc = (unsigned char*)lpDIBBits + lLineBytes * i + j;

        //该点如果为黑点
```

```
            if (*lpSrc == 0)
            {
                //设置下边界
                rectnew.bottom = i+1;

                //对 i 进行强制定义以跳出循环
                i=-1;
                //跳出循环
                break;
            }
        }
    }

    //将得到的新的准确的位置信息从后面插到链表 2 的尾上
    charRect2.push_back (rectnew);
}

//将链表 2 传递给链表 1
charRect1=charRect2;

//解除锁定
::GlobalUnlock(hDIB);

//将链表 1 返回
return charRect1;
}
```

为了使读者能够清楚地看出图像分割的结果，这里设计了一个函数 DrawFrame 用来在每个已经分割完毕的字符周围画边框，这个画框函数只会起到标识图像的作用，并不会对位图本身的内容造成改变。

这里，简要介绍一下 STL 模板中队列链表 deque 库的使用。要使用该库，首先要作如下声明：

```
#include <iostream>
#include <deque>
using namespace std;
typedef deque<CRect> CRectLink;
typedef deque<HDIB>  HDIBLink;
```

这样，以后用 CRectLink 就可以定义一个元素为 Crect 的链表了。HDIBLink 也是一个链表，不过其中的元素为位图句柄 HDIB。

假如用 CrectLink 定义了一个链表 a，那么：

a. push_back(rect)可以将一个矩形区域对象插入到链表后部；

a. front()可以读取链表头部的一个矩形对象（只是读取，不删除；

a. pop_front()可以删除链表头部的一个矩形对象；

a. empty()如果链表为空，则返回 1；否则，为 0。

至此相信读者对 STL 模板类中的 deque 已经有了一个基本的认识，该链表的使用十分方便。

下面介绍编写字符分割菜单处理事件的代码：

```
//图像预处理第 6 步：分割，并在分割出来的字符外面画框以标识
void CChildView::OnImgprcDivide()
{
 //对位图按照字符进行分割，返回的是存有分割后的每个含字符区域的链表
 m_charRect=CharSegment(m_hDIB);

 //在屏幕上显示位图
 CDC* pDC=GetDC();
 DisplayDIB(pDC,m_hDIB);

 //在分割好的字符周围画框标识
 DrawFrame(pDC,m_hDIB,m_charRect,2,RGB(20,60,200));
}
```

进行字符分割后并画上边框的图像如图 11-10 所示。

0 1 2 3 4 5 6 7 8 9

图 11-10　经过字符分割后并画上边框的图像

（7）图像的归一化处理

因为扫描进来的图像中字符大小存在较大的差异，而相对来说，统一尺寸的字符识别的标准性更强，准确率自然也更高，标准化图像就是要把原来各不相同的字符统一到同一尺寸，在本系统中是统一到同一高度，然后根据高度来调整字符的宽度。具体算法如下：先得到原来字符的高度，并与系统要求的高度做比较，得出要变换的系数，然后根据得到的系数求得变换后应有得宽度。在得到宽度高度之后，把新图像里面的点按照插值的方法映射到原图像中。

图像的标准归一化处理由函数 StdDIBbyRect 来实现，代码如下：

```
/*******************************************************************
 *
 * 函数名称：
 *     StdDIBbyRect()
 *
 * 参数：
 *    HDIB  hDIB          一图像的句柄
 *    int   tarWidth       一标准化的宽度
 *    int   tarHeight      一标准化的高度
 *
 * 返回值：
 *        无
 *
```

```
*  功能:
*      将经过分割的字符，进行缩放处理使他们的宽和高一直，以方便特征的提取
*
*  说明:
*      函数中用到了每个字符的位置信息，所以必须在执行完分割操作之后才能执行标准化操作
*
*********************************************************************/
void StdDIBbyRect(HDIB hDIB, int tarWidth, int tarHeight)
{

 //指向图像的指针
 BYTE* lpDIB=(BYTE*)::GlobalLock ((HGLOBAL)hDIB);

 //指向像素起始位置的指针
 BYTE* lpDIBBits=(BYTE*)::FindDIBBits ((char*)lpDIB);

 //指向像素的指针
 BYTE* lpSrc;

 //获取图像的宽度
 LONG lWidth=::DIBWidth ((char*)lpDIB);

 //获取图像的高度
 LONG lHeight=::DIBHeight ((char*)lpDIB);

 // 循环变量
 int i;
 int j;

 // 图像每行的字节数
 LONG    lLineBytes = WIDTHBYTES(lWidth * 8);

 //宽度、高度方向上的缩放因子
 double wscale,hscale;

 //开辟一块临时缓存区,来存放变化后的图像信息
 LPSTR lpNewDIBBits;

 LPSTR lpDst;

 //缓存区的大小和原图像的数据区大小一样
 HLOCAL nNewDIBBits=LocalAlloc(LHND,lLineBytes*lHeight);

 //指向缓存区开始位置的指针
```

```
lpNewDIBBits=(char*)LocalLock(nNewDIBBits);

//指向缓存内信息的指针
lpDst=(char*)lpNewDIBBits;

//将缓存区的内容赋初始值
memset(lpDst,(BYTE)255,lLineBytes*lHeight);

//进行映射操作的坐标变量
int i_src,j_src;

//存放字符位置信息的 Crect 对象
CRect rect;
CRect rectnew;

//先清空一个新的矩形区域链表以便存储标准化后的矩形区域链表
m_charRectCopy.clear ();

//从头到尾逐个扫描各个结点
while(!m_charRect.empty())
{
    //从表头上得到一个矩形
    rect= m_charRect.front();

    //从链表头上面删掉一个
    m_charRect.pop_front();

    //计算缩放因子

    //横坐标方向的缩放因子
    wscale=(double)tarWidth/rect.Width ();

    //纵坐标方向的缩放因子
    hscale=(double)tarHeight/rect.Height ();

    //计算标准化矩形

    //上边界
    rectnew.top =rect.top ;

    //下边界
    rectnew.bottom =rect.top +tarHeight;

    //左边界
```

```
    rectnew.left =rect.left ;

    //右边界
    rectnew.right =rectnew.left +tarWidth;

    //将原矩形框内的像素映射到新的矩形框内
    for(i=rectnew.top ;i<rectnew.bottom ;i++)
    {
        for(j=rectnew.left ;j<rectnew.right ;j++)
        {

            //计算映射坐标
            i_src=rectnew.top +int((i-rectnew.top )/hscale);
            j_src=rectnew.left +int((j-rectnew.left )/wscale);

            //将相对应的像素点进行映射操作
            lpSrc=(unsigned char *)lpDIBBits + lLineBytes *  i_src + j_src;
            lpDst = (char *)lpNewDIBBits + lLineBytes * i + j;
            *lpDst=*lpSrc;
        }

        //将标准化后的矩形区域插入新的链表
        m_charRectCopy.push_back (rectnew);

    }
}

//存储标准化后新的 rect 区域
m_charRect=m_charRectCopy;

//将缓存区的内容拷贝到图像的数据区内
memcpy(lpDIBBits,lpNewDIBBits,lLineBytes*lHeight);

//解除锁定
::GlobalUnlock ((HGLOBAL)hDIB);
}
```

图像标准归一化的高度和宽度信息可以通过一个对话框由读者自行输入，但是为了以后特征提取的时候处理方便，进行 BP 网络训练的时候缩短训练的时间，标准化的宽度和高度不要设置的过大。建议去归一化宽度为 8，高度为 16。

下面就是利用对话框来输入标准化宽度和高度的菜单事件的代码：

```
void CChildView::OnInputGuiyihuaInfo()
{
//创建对话框
CINPUT1 input;
```

```
//为变量赋默认初始值
input.w =8;
input.h =16;

//提示用户输入归一化的尺寸信息
if(input.DoModal ()!=IDOK) return;

//获取用户输入的信息
w_sample=input.w;
h_sample=input.h;

//设置标志位表明已经输入过归一化的尺寸信息了
gyhinfoinput=true;
}
```

归一化信息输入对话框的执行效果如图11-11所示。

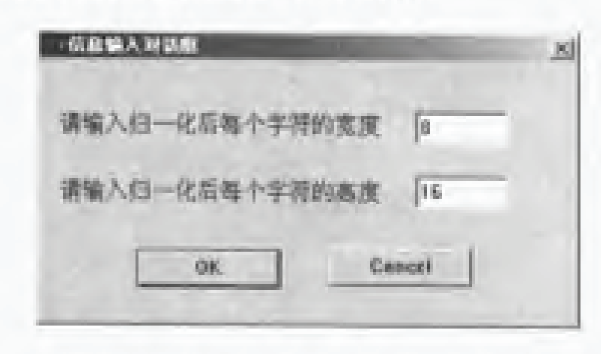


图11-11　利用对话框输入归一化尺寸信息

下面是对图像进行标准归一化的菜单处理事件响应代码：

```
//图像预处理第7步：标准归一化
//将分割出来的各个不同宽、高的数字字符宽、高归一化以便于后续操作
void CChildView::OnImgprcStandarize()
{
    //调用归一化处理函数
  StdDIBbyRect(m_hDIB,w_sample,h_sample);

  //在屏幕上显示位图
  CDC* pDC=GetDC();
  DisplayDIB(pDC,m_hDIB);

    //为归一化后的字符画上边框
  DrawFrame(pDC,m_hDIB,m_charRect,2,RGB(21,255,25));
}
```

归一化执行效果如图 11-12 所示。

图 11-12　对图像进行归一化处理

（8）图像的紧缩重排

经过标准归一化处理后的各个字符在图像中的位置不定，要它进行特征提取时处理起来比较麻烦，所以要把归一化后的字符进行紧缩重排，以形成新的位图句柄，以方便下一步的特征提取的操作。紧缩重排由函数 AutoAlign()来实现，代码如下：

```
/*********************************************
*
*  函数名称：
*  AutoAlign()
*
*  参数：
*    HDIB   hDIB          —原图像的句柄
*
*  返回值
*    HDIB                —紧缩排列后的新图像的句柄
*
*  功能：
*      将经过了标准化处理的字符进行规整的排列，以方便下一步的处理
*
*  说明：
*      紧缩排列的操作必须在标准化操作之后进行
*
*******************************************************/

HDIB AutoAlign(HDIB hDIB)
{

 //指向图像的指针
 BYTE* lpDIB=(BYTE*)::GlobalLock ((HGLOBAL)hDIB);

 //指向像素起始位置的指针
 BYTE* lpDIBBits=(BYTE*)::FindDIBBits ((char*)lpDIB);

 //指向像素的指针
 BYTE* lpSrc;

 //获取图像的宽度
 LONG lWidth=::DIBWidth ((char*)lpDIB);
```

```
//获取图像的高度
LONG lHeight=::DIBHeight ((char*)lpDIB);

//获取标准化的宽度
int w=m_charRect.front ().Width() ;

//获取标准化的高度
int h=m_charRect.front ().Height() ;

//建立一个新的图像正好能够将标准化的字符并排放置
HDIB hNewDIB=::NewDIB (digicount*w,h,8);

//指向新的图像的指针
BYTE* lpNewDIB=(BYTE*) ::GlobalLock((HGLOBAL)hNewDIB);

//指向像素起始位置的指针
BYTE* lpNewDIBBits=(BYTE*)::FindDIBBits((char*)lpNewDIB);

//指向像素的指针
BYTE* lpDst=lpNewDIBBits;

//计算原图像每行的字节数
LONG lLineBytes=(lWidth+3)/4*4;

//计算新图像每行的字节数
LONG lLineBytesnew =(digicount*w+3)/4*4;

//将新的图像初始化为白色
memset(lpDst,(BYTE)255,lLineBytesnew * h);

//映射操作的坐标变量
int i_src,j_src;

//循环变量
int i,j;

//统计字符个数的变量
int counts=0;

//存放位置信息的结构体
CRect rect,rectnew;

//清空一个新的链表来存放新的字符位置信息
```

```
m_charRectCopy.clear ();

//从头至尾逐个扫描原链表的各个结点
while(!m_charRect.empty() )
{
    //从表头上得到一个矩形框
    rect=m_charRect.front ();

    //将这个矩形框从链表上删除
    m_charRect.pop_front ();

    //计算新的矩形框的位置信息

    //左边界
    rectnew.left =counts*w;

    //右边界
    rectnew.right =(counts+1)*w;

    //上边界
    rectnew.top =0;

    //下边界
    rectnew.bottom =h;

    //将获得的新的矩形框插入到新的链表中
    m_charRectCopy.push_back (rectnew);

    //将原矩形框内的像素映射到新的矩形框中
    for(i=0;i<h;i++)
    {
        for(j=counts*w;j<(counts+1)*w;j++)
        {

            //计算映射坐标
            i_src=rect.top +i;
            j_src=rect.left +j-counts*w;

            //进行像素的映射
            lpSrc=(BYTE *)lpDIBBits + lLineBytes *  i_src + j_src;
            lpDst=(BYTE *)lpNewDIBBits + lLineBytesnew * i + j;
            *lpDst=*lpSrc;
        }
    }
```

```
        //字符个数加 1
        counts++;
    }

    //将获得的新的链表复制到原链表中，以方便下一次的调用
    m_charRect=m_charRectCopy;

    //解除锁定
    ::GlobalUnlock (hDIB);

    ::GlobalUnlock (hNewDIB);
    return hNewDIB;
}
```

下面来添加紧缩重排的菜单处理事件代码：

```
//图像预处理第 8 步：紧缩重排已经分割完毕的数字字符，并形成新的位图句柄
void CChildView::OnImgprcShrinkAlign()
{
    //调用函数实现紧缩重排功能
    m_hDIB=AutoAlign(m_hDIB);

    //在屏幕上显示图像
    CDC* pDC=GetDC();
    DisplayDIB(pDC,m_hDIB);

    //为重排后的字符化上边框
    DrawFrame(pDC,m_hDIB,m_charRect,1,RGB(252,115,27));
}
```

程序执行效果如图 11-13 所示。

0123456789

图 11-13　图像紧缩重排后的效果

至此已经基本完成图像的预处理。此外，还可以采用其他方法以更好地达到预处理的效果，比如可以对字符进行光滑处理等。但考虑到光滑对该识别系统的改进很细微，笔者没有使用光滑算法，也没有提供源代码。读者感兴趣的可以自行编写、测试。

（9）特征提取

经过上面一系列的变换，原来，大小不同，分布不规律的各个字符变成了一个个大小相同，排列整齐的字符。下面就要从被分割归一处理完毕的字符中，提取最能体现这个字符特点的特征向量。将提取出训练样本中的特征向量代入 BP 网络之中就可以对网络进行训练，提取出待识别的样本中的特征相量代入到训练好的 BP 网络中，就可以对字符进行识别。特征向量的提取方法多种多样，有逐像素特征提取发，骨架特征提取法，垂直方向数据统计特征提取法，13

点特征提取法，弧度梯度特征提取法等很多种方法，根据具体情况的不同我们可以来选择不同的方法。

下面笔者提供了几种简单的特征提取方法：

① 逐像素特征提取法

这是一种最简单的特征提取方法，对图像进行逐行逐列的扫描当遇到黑色像素时取其特征值为 1，遇到白色像素时取其特征值为 0，这样当扫描结束以后就形成了一个维数与图像中像素点的个数相同的特征向量矩阵。

这种算法可以由函数 code 实现：

```
/************************************
* 函数名称 code()
*
* 参量:
*   BYTE* lpDIBBits    —指向输入图像的像素其实位置的指针
*   int num           —图片中样本的个数
*   LONG lLineByte     —输入图片每行的字节数
*   LONG lSwidth       —预处理时归一化的宽度
*   LONG lSheight      —预处理时归一化的长度
*
* 返回值:
*   double**           —特征向量矩阵（二维的）
*
*  函数功能 :
*       对于输入样本提取特征向量，在这里把归一化样本的
*         每一个像素都作为特征提取出来
**************************************/

double** code (BYTE* lpDIBBits,int num, LONG lLineByte,LONG lSwidth,LONG
lSheight)
{
 //循环变量
 int i,j,k;
 BYTE* lpSrc;

 //  建立保存特征向量的二维数组
 double **data;

 // 为这个数组申请二维存储空间
 data = alloc_2d_dbl(num,lSwidth*lSheight);

 // 将归一化的样本的每个像素作为一个特征点提取出来

 //逐个数据扫描
 for(k=0;k<num;k++)
 {
```

```
    //对每个数据逐行扫描
    for(i=0;i<lSheight;i++)
    {
        //对每个数据逐列扫描
        for(j=k*lSwidth;j<(k+1)*lSwidth;j++)
        {

            // 指向图像第 i 行第 j 列个像素的指针
            lpSrc = lpDIBBits + i*lLineByte + j;

            //如果这个像素是黑色的
            if(*(lpSrc)==0)

                //将特征向量的相应位置填 1
                data[k][i*lSwidth+j-k*lSwidth]=1;

            //如果这个像素是其他的颜色
            if(*(lpSrc)!=0)

                //将特征向量的相应位置填 0
                data[k][i*lSwidth+j-k*lSwidth]=0;
        }
    }
}

//返回特征向量矩阵
return(data);
}
```

这种特征提取方法的特点是算法简单，运算速度快，可以使 BP 网络很快地收敛，训练效果好，缺点是适应性不强。但可以通过加大训练样本数目的方法来增强其适应性。

② 骨架特征提取法

两幅图像由于它们的线条的粗细不同，使得两幅图像差别很大，但是将它们的线条进行细化以后，统一到相同的宽度，如一个像素宽时，这时两幅图像的差距就不那么明显。利用图形的骨架作为特征来进行数码识别，就使得识别有了一定的适应性。一般使用细化的方法来提取骨架，细化的算法又很多如 Hilditch 算法，Rosenfeld 算法等，下面介绍这两种算法的代码。

Hilditch 细化算法：

```
/*************************************************************
* 函数名称:
*     ThinnerHilditch
*
* 参数:
*   void*     image              一二值化图像矩阵前景色为 1 背景色为 0
*   unsigned  longlx             一图像的宽度
*   unsigned  longly             一图像的高度
```

```
*
* 返回值
*       无
*
*函数功能:
*       对输入的图像进行细化，输出细化后的图像
*************************************************************

void ThinnerHilditch(void *image, unsigned long lx, unsigned long ly)
{
    char *f, *g;
    char n[10];
    unsigned int counter;
    short k, shori, xx, nrn;
    unsigned long i, j;
    long kk, kk11, kk12, kk13, kk21, kk22, kk23, kk31, kk32, kk33, size;
    size = (long)lx * (long)ly;
    g = (char *)malloc(size);

    if(g == NULL)
    {
       // printf("error in allocating memory!\n");
        return;
    }

    f = (char *)image;
    for(i=0; i<lx; i++)
    {
        for(j=0; j<ly; j++)
        {
            kk=i*ly+j;
            if(f[kk]!=0)
            {
                f[kk]=1;
                g[kk]=f[kk];
            }
        }
    }

    counter = 1;

    do
    {
        //printf("%4d*",counter);
```

```
counter++;
shori = 0;

for(i=0; i<lx; i++)
{
    for(j=0; j<ly; j++)
    {
        kk = i*ly+j;
        if(f[kk]<0)
            f[kk] = 0;
        g[kk]= f[kk];
    }
}

for(i=1; i<lx-1; i++)
{
    for(j=1; j<ly-1; j++)
    {
        kk=i*ly+j;

        if(f[kk]!=1)
            continue;

        kk11 = (i-1)*ly+j-1;
        kk12 = kk11 + 1;
        kk13 = kk12 + 1;
        kk21 = i*ly+j-1;
        kk22 = kk21 + 1;
        kk23 = kk22 + 1;
        kk31 = (i+1)*ly+j-1;
        kk32 = kk31 + 1;
        kk33 = kk32 + 1;

        if((g[kk12]&&g[kk21]&&g[kk23]&&g[kk32])!=0)
            continue;

        nrn = g[kk11] + g[kk12] + g[kk13] + g[kk21] + g[kk23] +
            g[kk31] + g[kk32] + g[kk33];

        if(nrn <= 1)
        {
            f[kk22] = 2;
            continue;
        }
```

```
n[4] = f[kk11];
n[3] = f[kk12];
n[2] = f[kk13];
n[5] = f[kk21];
n[1] = f[kk23];
n[6] = f[kk31];
n[7] = f[kk32];
n[8] = f[kk33];
n[9] = n[1];
xx = 0;

for(k=1; k<8; k=k+2)
{
    if((!n[k])&&(n[k+1]||n[k+2]))
        xx++;
}

if(xx!=1)
{
    f[kk22] = 2;
    continue;
}

if(f[kk12] == -1)
{
    f[kk12] = 0;
    n[3] = 0;
    xx = 0;

    for(k=1; k<8; k=k+2)
    {
        if((!n[k])&&(n[k+1]||n[k+2]))
            xx++;
    }

    if(xx != 1)
    {
        f[kk12] = -1;
        continue;
    }

    f[kk12] = -1;
    n[3] = -1;
```

```
            }

            if(f[kk21]!=-1)
            {
                f[kk22] = -1;
                shori = 1;
                continue;
            }

            f[kk21] = 0;
            n[5] = 0;
            xx = 0;

            for(k=1; k<8; k=k+2)
            {
                if((!n[k])&&(n[k+1]||n[k+2]))
                {
                    xx++;
                }
            }

            if(xx == 1)
            {
                f[kk21] = -1;
                f[kk22] = -1;
                shori =1;
            }
            else
                f[kk21] = -1;
        }
    }
  }while(shori);

  free(g);
}
```

ThinnerRosenfeld 细化算法，代码如下：

```
/*************************************************************
* 函数名称:
*    ThinnerRosenfeld
*
* 参数:
*   void*     image              一二值化图像矩阵前景色为 1 背景色为 0
*   unsigned  longlx             一图像的宽度
*   unsigned  longly             一图像的高度
```

```
*
* 返回值
*       无
*
*函数功能:
*       对输入的图像进行细化，输出细化后的图像
*************************************************************
void ThinnerRosenfeld(void *image, unsigned long lx, unsigned long ly)
{
    char *f, *g;
    char n[10];
    char a[5] = {0, -1, 1, 0, 0};
    char b[5] = {0, 0, 0, 1, -1};
    char nrnd, cond, n48, n26, n24, n46, n68, n82, n123, n345, n567, n781;
    short k, shori;
    unsigned long i, j;
    long ii, jj, kk, kk1, kk2, kk3, size;
    size = (long)lx * (long)ly;

    g = (char *)malloc(size);
    if(g==NULL)
    {
        printf("error in alocating mmeory!\n");
        return;
    }

    f = (char *)image;
    for(kk=0l; kk<size; kk++)
    {
        g[kk] = f[kk];
    }

    do
    {
        shori = 0;
        for(k=1; k<=4; k++)
        {
            for(i=1; i<lx-1; i++)
            {
                ii = i + a[k];

                for(j=1; j<ly-1; j++)
                {
                    kk = i*ly + j;
```

```
if(!f[kk])
    continue;

jj = j + b[k];
kk1 = ii*ly + jj;

if(f[kk1])
    continue;

kk1 = kk - ly -1;
kk2 = kk1 + 1;
kk3 = kk2 + 1;
n[3] = f[kk1];
n[2] = f[kk2];
n[1] = f[kk3];
kk1 = kk - 1;
kk3 = kk + 1;
n[4] = f[kk1];
n[8] = f[kk3];
kk1 = kk + ly - 1;
kk2 = kk1 + 1;
kk3 = kk2 + 1;
n[5] = f[kk1];
n[6] = f[kk2];
n[7] = f[kk3];

nrnd = n[1] + n[2] + n[3] + n[4]
    +n[5] + n[6] + n[7] + n[8];
if(nrnd<=1)
    continue;

cond = 0;
n48 = n[4] + n[8];
n26 = n[2] + n[6];
n24 = n[2] + n[4];
n46 = n[4] + n[6];
n68 = n[6] + n[8];
n82 = n[8] + n[2];
n123 = n[1] + n[2] + n[3];
n345 = n[3] + n[4] + n[5];
n567 = n[5] + n[6] + n[7];
n781 = n[7] + n[8] + n[1];
```

```
if(n[2]==1 && n48==0 && n567>0)
{
    if(!cond)
        continue;
    g[kk] = 0;
    shori = 1;
    continue;
}

if(n[6]==1 && n48==0 && n123>0)
{
    if(!cond)
        continue;
    g[kk] = 0;
    shori = 1;
    continue;
}

if(n[8]==1 && n26==0 && n345>0)
{
    if(!cond)
        continue;
    g[kk] = 0;
    shori = 1;
    continue;
}

if(n[4]==1 && n26==0 && n781>0)
{
    if(!cond)
        continue;
    g[kk] = 0;
    shori = 1;
    continue;
}

if(n[5]==1 && n46==0)
{
    if(!cond)
        continue;
    g[kk] = 0;
    shori = 1;
    continue;
}
```

```
            if(n[7]==1 && n68==0)
            {
                if(!cond)
                    continue;
                g[kk] = 0;
                shori = 1;
                continue;
            }

            if(n[1]==1 && n82==0)
            {
                if(!cond)
                    continue;
                g[kk] = 0;
                shori = 1;
                continue;
            }

            if(n[3]==1 && n24==0)
            {
                if(!cond)
                    continue;
                g[kk] = 0;
                shori = 1;
                continue;
            }

            cond = 1;
            if(!cond)
                continue;
            g[kk] = 0;
            shori = 1;
        }
    }

    for(i=0; i<lx; i++)
    {
        for(j=0; j<ly; j++)
        {
            kk = i*ly + j;
            f[kk] = g[kk];
        }
    }
```

```
        }
    }while(shori);

    free(g);
}
```

对图像进行骨架提取的效果如图 11-14 所示。

原始图像 0123456789

细化以后 0123456789

图 11-14 细化效果图

对经过细化的图相利用 EveryPixel 函数进行处理就可以得到细化后图像的特征向量矩阵。

骨架特征提取的方法对于线条粗细不同的数码有一定的适应性，但是图像一旦出现偏移就难以识别。

③ 垂直方向数据统计特征提取法

这种特征提取方法的算法就是自左向右对图像进行逐列的扫描，统计每列黑色像素的个数，然后自上而下逐行扫描，统计每行的黑色像素的个数，将统计结果作为字符的特征向量，如果字符的宽度为 w，长度为 h，则特征向量的维数是 $w+h$。这种特征提取方法可以由函数 VerticalCode 来实现。源代码如下：

```
/*************************************************************
* 函数名称 VerticalCode()
*
* 参量:
*   BYTE* lpDIBBits    —指向输入图像的像素起始位置的指针
*   int  num          —图片中样本的个数
*   LONG lLineByte     —输入图片每行的字节数
*   LONG lSwidth       —预处理时归一化的宽度
*   LONG lSheight      —预处理时归一化的长度
*
* 返回值:
*   double**           —特征向量矩阵
*
*  函数功能 :
*       对于输入样本提取特征向量，在这里把归一化样本的
*        水平和竖直方向的统计特征作为特征提取出来
*****************************************************************/

double**  VerticalCode(BYTE*  lpDIBBits,int  num,  LONG  lLineByte,LONG
```

```
lSwidth,LONG lSheight)
{
    //循环变量
    int i,j,k;
    BYTE* lpSrc;

    //统计变量
    int sum;

    //  建立保存特征向量的二维数组
    double **data;

    // 为这个数组申请二维存储空间
    data = alloc_2d_dbl(num,lSwidth+lSheight);

    // 将归一化的样本的每个像素作为一个特征点提取出来

    //逐个数据扫描
    for(k=0;k<num;k++)
    {
        //统计每行的像素点个数
        for(i=0;i<lSheight;i++)
        {
            //对统计变量初始化
            sum=0;

            //对每个数据逐列扫描
            for(j=k*lSwidth;j<(k+1)*lSwidth;j++)
            {
                // 指向图像第 i 行第 j 列个像素的指针
                lpSrc = lpDIBBits + i*lLineByte + j;

                //如果这个像素是黑色的
                if(*(lpSrc)==0)

                    //统计变量加 1
                    sum++;
            }
            data[k][i]=sum;
        }

        //统计每列的像素点个数
        for(j=k*lSwidth;j<(k+1)*lSwidth;j++)
        {
```

```
            //对统计变量初始化
            sum=0;

            //对每个数据逐行扫描
            for(i=0;i<lSheight;i++)
            {
                // 指向图像第 i 行第 j 列个像素的指针
                lpSrc = lpDIBBits + i*lLineByte + j;

                //如果这个像素是黑色的
                if(*(lpSrc)==0)

                    //统计变量加 1
                    sum++;
            }
            data[k][j-k*lSwidth+lSheight]=sum;
        }

    }

    //返回特征向量矩阵
    return(data);
}
```

④ 13 特征点提取方法

上述的特征点提取方法多少都存在有适应性不强的特点，当字符存在倾斜和偏移时都会对识别产生误差，下面来介绍一种适应性较好的 13 点特征提取方法，即从每个字符中提取 13 个特征点。

首先把字符平均分成 8 份统计每一份内黑色像素点的个数作为 8 个特征，如图 11-15 所示。(这里以字符“3”举例。) 分别统计这 8 个区域中的黑像素的数目，可以得到 8 个特征。



图 11-15　13 特征提取法

然后统计水平方向中间两列和竖直方向中间两列的黑色像素点的个数作为 4 个特征，最后统计所有黑色像素点的个数作为第 13 个特征。也就是说，画 4 道线，统计线穿过的黑像素的数目。可以得到 4 个特征示意图如图 11-16 所示。

3 3

图 11-16　13 特征提取法

最后，将字符图像的全部黑色像素的数目的总和作为一个特征。总共即得到 13 个特征。该算法可由函数 TZTQ_13 来实现，其完整代码如下：

```
/**********************************************************************
*
*函数名称：
*    TZTQ_13
*参数：
*    HDIB hDIB                 -待提取特征的位图的句柄
*    int num                   -字符的数目
*    int dim                   -提取特征的维数。这里固定为 13
*说明：
*    图像分为 8 块，作为 8 个特征；像素总数作为一个特征；水平切割过去两条线，得*到两个特征；垂直的两个，总共得到 13 个特征
*
**********************************************************************/
double * * TZTQ_13(HDIB hDIB,int num,int dim)
{
 int i,j,k,m;

 //分配一个内存空间并得到二维指针
 double * * tezheng=alloc_2d_dbl(num,dim);

 //锁定图像句柄并获取其指针
 BYTE* lpDIB=(BYTE*)::GlobalLock ((HGLOBAL)hDIB);

 //取得图像像素数据区的起始地址
 BYTE* lpDIBBits=(BYTE*)::FindDIBBits((char*)lpDIB);
 BYTE* lpSrc;

 //获取图像高度
 LONG lHeight=::DIBHeight ((char*)lpDIB);

 //获取图像宽度
 LONG lWidth=::DIBWidth ((char*)lpDIB);

 LONG width=lWidth/num;
 //每行的字节数
 LONG lLineBytes = WIDTHBYTES(lWidth * 8);
```

```
int b;

//存储临时的特征
double * tz=new double[dim];

for(k=0;k<num;k++)
{
    for(i=0;i<dim;i++) tz[i]=0;

    //提取前 8 个特征
    for(m=0;m<8;m++)
    {   for(i=int(m/2)*8;i<(int(m/2)+1)*8;i++)
    for(j=m%2*8+k*width;j<(m%2+1)*8+k*width;j++)
    {   lpSrc=(unsigned char*)lpDIBBits + lLineBytes *  i + j;
    b=(*lpSrc==255)?0:1;
    tz[m]+=b;
    }
    }

    //提取第 9 个特征-总像素值
    for(i=0;i<lHeight;i++)
        for(j=k*width;j<(k+1)*width;j++)
        {   lpSrc=(unsigned char*)lpDIBBits + lLineBytes *  i + j;
        b=(*lpSrc==255)?0:1;
        tz[8]+=b;
        }

        //提取第 10、11 个特征-水平扫描切割
        i=int(lHeight*1/3);
        for(j=k*width;j<(k+1)*width;j++)
        {   lpSrc=(unsigned char*)lpDIBBits + lLineBytes *  i + j;
        b=(*lpSrc==255)?0:1;
        tz[9]+=b;
        }
        //
        i=int(lHeight*2/3);
        for(j=k*width;j<(k+1)*width;j++)
        {   lpSrc=(unsigned char*)lpDIBBits + lLineBytes *  i + j;
        b=(*lpSrc==255)?0:1;
        tz[10]+=b;
        }

        //提取第 12、13 个特征-垂直扫描切割
```

```
        j=int(k*width+width*1/3);
        for(i=0;i<lHeight;i++)
        {    lpSrc=(unsigned char*)lpDIBBits + lLineBytes *  i + j;
        b=(*lpSrc==255)?0:1;
        tz[11]+=b;
        }

        j=int(k*width+width*2/3);
        for(i=0;i<lHeight;i++)
        {    lpSrc=(unsigned char*)lpDIBBits + lLineBytes *  i + j;
        b=(*lpSrc==255)?0:1;
        tz[12]+=b;
        }

        //存储特征
        for(i=0;i<dim;i++)
            tezheng[k][i]=tz[i];
}

::GlobalUnlock ((HGLOBAL)hDIB);
//返回特征向量矩阵的指针
return tezheng;
}
```

13 特征提取法有着极好的适应性，但是由于特征点的数目太少所以在样本训练的时候比较难收敛。

以上就是几种基本的特征向量提取方法，还有梯度统计，弧度统计等其他的特征向量提取方法，请读者自行尝试。

另外，还有一种效率极高的特征提取方法一角点提取方法。角点提取的方法目前一般分为基于灰度的角点提取和基于弧度的角点提取。基于弧度的角点提取要求首先要做边缘提取。比较成熟的边缘提取算法有 Canny 算法等。角点提取算法较复杂，而且用到的很多技术也较专业化，限于篇幅，这里笔者就不在赘述了。

在本文中，经多次尝试，最终选用的是第一种特征提取方法。读者可以试用其他几种方法，从训练时间和识别率上加以对比。

3. 其他相关的重要图像处理技术

在这里，单独拿出一小节，来给读者介绍各种滤波去噪技术。文中前面讲到的离散点去噪的办法之外，还有很多经典的滤波方案可以选择。而且面对不同的含数字的图片，读者可以自行选用更加适合的滤波方法。

这里向读者介绍的技术主要包括平滑技术、中值滤波技术和直方图均衡技术。

（1）平滑技术（均值平滑与高斯平滑）

一些打印的文本例如传真、复制材料和历史记录等，字符由于分辨率的不足有了失真的形状，并且许多的字符断裂了，尽管人为的填充这些裂缝在视觉上并不困难，但是对于机器识别系统来说要处理这些断裂的字符却很困难。这就需要利用低通滤波的方法来对图像进行处理，

使断裂的图像产生平滑、柔和的外观。所以二维图像的低通滤波又叫图像的平滑。

图像平滑的效果图如图 11-17 所示。

原图

均值平滑后

图 11-17　图像平滑处理

可以看到，图像平滑后和原图相比柔和一些（也模糊一些）。其实实现起来很简单，将原图中的每一点的灰度和它周围 8 个点的灰度相加，然后除以 9，作为新图中对应点的灰度，就能实现上面的效果。这么做并非瞎蒙，而是有其道理的。大概想一想，也很容易明白。举个例子，就像和面一样，先在中间加点水，然后不断把周围的面和进来，搅拌几次，面就均匀了。

用信号处理的理论来解释，这种做法实现的是一种简单的低通滤波器（low pass filter）。在灰度连续变化的图像中，如果出现了与相邻像素的灰度相差很大的点，比如说一片暗区中突然出现了一个亮点，人眼能很容易觉察到。就象看老电影时，由于胶片太旧，屏幕上经常会出现一些亮斑。这种情况被认为是一种噪声。灰度突变在频域中代表了一种高频分量，低通滤波器的作用就是滤掉高频分量，从而达到减少图像噪声的目的。

为了方便的叙述上面所说的“将原图中的每一点的灰度和它周围 8 个点的灰度相加，然后除以 9，作为新图中对应点的灰度”这一操作，用如下的表示方法：

$$\frac{1}{9}\begin{bmatrix}1, & 1, & 1\\ 1, & 1^*, & 1\\ 1, & 1, & 1\end{bmatrix}$$

这种表示方法有点像矩阵，我们称其为模板(template)。中间的黑点表示中心元素（此处用*表示），即，用哪个元素做为处理后的元素。例如(2*,1)表示将自身的 2 倍加上右边的元素作为新值，而(2,1*)表示将自身加上左边元素的 2 倍作为新值。通常，模板不允许移出边界，所以结果图像会比原图小，加入模板为：

$$\begin{bmatrix}1^*, & 0\\ 0 & 1\end{bmatrix}$$

原图为：

$$\begin{bmatrix}1, & 1, & 1, & 1, & 1\\ 2, & 2, & 2, & 2, & 2\\ 3, & 3, & 3, & 3, & 3\\ 4, & 4, & 4, & 4, & 4\end{bmatrix}$$

则经过模板操作后图像为：

$$\begin{bmatrix} 3, & 3, & 3, & 3, & x \\ 5, & 5, & 5, & 5, & x \\ 7, & 7, & 7, & 7, & x \\ x, & x, & x, & x, & x \end{bmatrix}$$

数字代表灰度，x 表示边界上无法进行模板操作的点，通常的做法是复制原图的灰度，不进行任何处理。模板操作实现了一种邻域运算（Neighborhood Operation），即某个像素点的结果不仅和本像素灰度有关，而且和其邻域点的值有关。在以后介绍的细化算法中，我们还将接触到邻域运算。模板运算的数学涵义是一种卷积（或互相关）运算，不需要知道卷积的确切概念，有这么一个概念就可以了。

平滑模板的思想是通过一点和周围 8 个点的平均来去除突然变化的点，从而滤掉一定的噪声，其代价是图像有一定程度的模糊。上面提到的模板（1），就是一种平滑模板，称之为 Box 模板。Box 模板虽然考虑了邻域点的作用，但并没有考虑各点位置的影响，对于所有的 9 个点都一视同仁，所以平滑的效果并不理想。实际上可以想象，离某点越近的点对该点的影响应该越大，为此，我们引入了加权系数，将原来的模板改造成：

$$\frac{1}{16}\begin{bmatrix} 1, & 2, & 1 \\ 2, & 4^*, & 2 \\ 1, & 2, & 1 \end{bmatrix}$$

如图 11-18 所示为原始均值平滑与高斯平滑的区别。

原始图像				均值平滑				高斯平滑			
2	2	2	2	2	2	2	2	2	2	2	2
3	2	3	3	3	3.11	3.22	3	3	3.00	3.06	3
4	6	4	5	4	4.33	4.56	5	4	4.44	4.5	5
5	5	6	6	5	5	6	6	5	5	6	6

图 11-18　平滑处理的区别

可以看到，原图中出现噪声的区域是第 2 行第 2 列和第 3 行第 2 列，灰度从 2 一下子跳到了 6，用 Box 模板处理后，灰度从 3.11 跳到 4.33；用高斯模板处理后，灰度从 3.跳到 4.56,都缓和了跳变的幅度，从这一点上看，两者都达到了平滑的目的。但是，原图中的第 3，第 4 行总的来说，灰度值是比较高的，经模板 1 处理后，第 3 行第 2 列元素的灰度变成了 4.33，与第 3，第 4 行的总体灰度相比偏小，另外，原图中第 3 行第 2 列元素的灰度为 6，第 3 行第 3 列元素的灰度为 4，变换后，后者 4.56 反而比前者 4.33 大了。而采用高斯模板没有出现这些问题，究其原因，就是因为它考虑了位置的影响。

下面给出了模板操作源代码，这里默认为模板的中心元素位于模板的中心。

```
/*************************************************************
*
*  函数名称：
*     Template:
*
*   参数：
*      HDIB    hDIB        一图像的句柄
*     double  *tem        一指向模板的指针
```

```
*      int  tem_w          一模板的宽度
*      int  tem_h          一模板的高度
*      double xishu         一模板的系数
*
*    功能:
*        对图像进行模板操作
*
*    说明:
*        为处理方便起见,模板的宽度和高度都应为奇数
*****************************************************************/
HDIB Template(HDIB hDIB,double * tem ,int tem_w,int tem_h,double xishu)
{

 //统计中间值
 double sum;

 //指向图像起始位置的指针
 BYTE *lpDIB=(BYTE*)::GlobalLock((HGLOBAL) hDIB);

 //指向像素起始位置的指针
 BYTE *pScrBuff =(BYTE*)::FindDIBBits((char*)lpDIB);

 //获取图像的颜色信息
 int numColors=(int) ::DIBNumColors((char *)lpDIB);

 //如果图像不是256色返回
 if (numColors!=256)
 {
     //解除锁定
     ::GlobalUnlock((HGLOBAL) hDIB);

     //返回
     return(hDIB);
 }

 //将指向图像像素起始位置的指针,赋值给指针变量
 BYTE* oldbuf = pScrBuff;

 //循环变量
 int i,j,m,n;

 int w, h, dw;

 //获取图像的宽度
```

```
w = (int) ::DIBWidth((char *)lpDIB);

//获取图像的高度
h = (int) ::DIBHeight((char *)lpDIB);

//计算图像每行的字节数
dw = (w+3)/4*4;

//建立一个和原图像大小相同的 25 色灰度位图
HDIB newhDIB=NewDIB(w,h,8);

//指向新的位图的指针
BYTE *newlpDIB=(BYTE*)::GlobalLock((HGLOBAL) newhDIB);

//指向新的位图的像素起始位置的指针
BYTE *destBuf = (BYTE*)FindDIBBits((char *)newlpDIB);

//将指向新图像像素起始位置的指针，赋值给指针变量
BYTE *newbuf=destBuf;

//对图像进行扫描

//行
for(i=0;i<h;i++)
{
    //列
    for(j=0;j<w;j++)
    {

        //为统计变量赋初始值
        sum=0;

        //对于图像的 4 个边框的像素保持原灰度不变
        if( j<((tem_w-1)/2) || j>(w-(tem_w+1)/2) || i<((tem_h-1)/2) || i>(h-
(tem_h+1)/2) )
            *(newbuf+i*dw+j)=*(oldbuf+i*dw+j);

        //对于其他的像素进行模板操作
        else
        {

            //将点 (i,j) 点作为模板的中心
            for(m=i-((tem_h-1)/2);m<=i+((tem_h-1)/2);m++)
            {
```

```
                for(n=j-((tem_w-1)/2);n<=j+((tem_w-1)/2);n++)

            //将以点（i，j）为中心，与模板大小相同的范围内的像素与模板对用位置的系数
                    //进行相乘并线形叠加
sum+=*(oldbuf+m*dw+n)* tem[(m-i+((tem_h-1)/2))*tem_w+n-j+((tem_w-1)/2)];

            }

            //将结果乘上总的模板系数
            sum=(int)sum*xishu;

            //计算绝对值
            sum = fabs(sum);

            //如果小于 0，强制赋值为 0
            if(sum<0)
                sum=0;

            //如果大于 255，强制赋值为 255
            if(sum>255)
                sum=255;

            //将计算的结果放到新的位图的相应位置
            *(newbuf+i*dw+j)=sum;
        }
    }
}

//解除锁定
::GlobalUnlock((HGLOBAL)hDIB);

//返回新的位图的句柄
return(newhDIB);
}
```

添加均值平滑的菜单相应事件：

```
void CChildView::Onaver()
{
 // TODO: Add your command handler code here
 //设定模板参数
 double tem[9]={1,1,1,
     1,1,1,
     1,1,1};

 //设定模板系数
```

```
double  xishu = 0.111111;

//进行模板操作
m_hDIB =Template(m_hDIB,tem ,3,3, xishu);

//显示图像
CDC* pDC=GetDC();
DisplayDIB(pDC,m_hDIB);

}
```

添加高斯平滑的菜单相响应事件：

```
void CChildView::OnGass()
{
// TODO: Add your command handler code here

//设定模板参数
double tem[9]={1,2,1,
    2,4,2,
    1,2,1};

//设定模板系数
double  xishu = 0.0625;

//进行模板操作
m_hDIB =Template(m_hDIB,tem ,3,3, xishu);

//显示图像
CDC* pDC=GetDC();
DisplayDIB(pDC,m_hDIB);
}
```

（2）中值滤波技术

中值滤波也是一种典型的低通滤波器，它的目的是保护图像边缘的同时去除噪声。所谓中值滤波，就是指把以某点(x,y)为中心的小窗口内的所有像素的灰度按从大到小的顺序排列，将中间值作为(x,y)处的灰度值（若窗口中有偶数个像素，则取两个中间值的平均）。中值滤波是如何去除噪声的呢？先看下面的例子，如图 11-19 所示。

原图

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 1 1 1 0 0
0 0 1 6 1 0 0
0 0 1 1 1 0 0
0 0 0 0 0 0 0
```

处理后的图

```
0 0 0 0 0
0 0 0 0 0
0 1 1 1 0
0 1 1 1 0
0 1 1 1 0
0 0 0 0 0
```

图 11-19　中值滤波

上图中左边是原图，数字代表该处的灰度。可以看出中间的 6 和周围的灰度相差很大，是一个噪声点。经过 3×1 窗口（即水平 3 个像素取中间值）的中值滤波，得到右边那幅图，可以看出，噪声点被去除了。

中值滤波的源代码如下：

```
/*******************************************************
*
*   函数名称:
*      Template:
*
*    参数:
*       HDIB    hDIB          一图像的句柄
*      int  tem_w           一模板的宽度
*      int  tem_h           一模板的高度
*
*    功能:
*       对图像进行中值
*
*    说明:
*       为处理方便起见，模板的宽度和高度都应为奇数
********************************************************/

HDIB MidFilter(HDIB hDIB,int tem_w,int tem_h)
{

 //统计中间值
 double mid;

 BYTE *temp=(BYTE*)malloc(tem_w*tem_h*sizeof(BYTE));

 //指向图像起始位置的指针
 BYTE *lpDIB=(BYTE*)::GlobalLock((HGLOBAL) hDIB);

 //指向像素起始位置的指针
 BYTE *pScrBuff =(BYTE*)::FindDIBBits((char*)lpDIB);

 //获取图像的颜色信息
 int numColors=(int) ::DIBNumColors((char *)lpDIB);

 //如果图像不是 256 色返回
 if (numColors!=256)
 {
      //解除锁定
      ::GlobalUnlock((HGLOBAL) hDIB);
```

```
    //返回
    return(hDIB);
}

//将指向图像像素起始位置的指针，赋值给指针变量
BYTE* oldbuf = pScrBuff;

//循环变量
int i,j,m,n;

int w, h, dw;

//获取图像的宽度
w = (int) ::DIBWidth((char *)lpDIB);

//获取图像的高度
h = (int) ::DIBHeight((char *)lpDIB);

//计算图像每行的字节数
dw = (w+3)/4*4;

//建立一个和原图像大小相同的 25 色灰度位图
HDIB newhDIB=NewDIB(w,h,8);

//指向新的位图的指针
BYTE *newlpDIB=(BYTE*)::GlobalLock((HGLOBAL) newhDIB);

//指向新的位图的像素起始位置的指针
BYTE *destBuf = (BYTE*)FindDIBBits((char *)newlpDIB);

//将指向新图像像素起始位置的指针，赋值给指针变量
BYTE *newbuf=destBuf;

//对图像进行扫描

//行
for(i=0;i<h;i++)
{
    //列
    for(j=0;j<w;j++)
    {

        //为统计变量赋初始值
```

```
        //对于图像的 4 个边框的像素保持原灰度不变
        if( j<((tem_w-1)/2) || j>(w-(tem_w+1)/2) || i<((tem_h-1)/2) || i>(h-
(tem_h+1)/2) )
            *(newbuf+i*dw+j)=*(oldbuf+i*dw+j);

        //对于其他的像素进行模板操作
        else
        {

            //将点 (i,j) 点作为模板的中心
            for(m=i-((tem_h-1)/2);m<=i+((tem_h-1)/2);m++)
            {
                for(n=j-((tem_w-1)/2);n<=j+((tem_w-1)/2);n++)

                //将以点 (i, j) 为中心，与模板大小相同的范围内的像素传递到模板矩阵中

    temp[(m-i+((tem_h-1)/2))*tem_w+n-j+((tem_w-1)/2)]=*(oldbuf+m*dw+n);

            }

            //利用气泡法计算中值
            for(m=0;m<tem_w*tem_h-1;m++)
            {

                for(n=0;n<tem_w*tem_h-m-1;n++)
                {
                    if(temp[n]>temp[n+1])
                        mid=temp[n];
                    temp[n]=temp[n+1];
                    temp[n+1]=mid;
                }

            }

            //将计算的结果放到新的位图的相应位置
            *(newbuf+i*dw+j)=temp[(tem_w*tem_h-1)/2];
        }
    }
}

//解除锁定
::GlobalUnlock((HGLOBAL)hDIB);
```

```
//返回新的位图的句柄
return(newhDIB);
}
```

添加中值滤波的菜单事件响应：

```
void CChildView::OnMid()
{

//进行中值滤波
m_hDIB =MidFilter(m_hDIB,3,3);

//显示图像
CDC* pDC=GetDC();
DisplayDIB(pDC,m_hDIB);
}
```

(3) 直方图均衡技术

图像直方图是图像处理中一种十分重要的图像分析工具，它描述了一幅图像的灰度级内容，任何一幅图像的直方图都包含了丰富的信息，它主要用在图像分割，图像灰度变换等处理过程中。从数学上来说图像直方图是图像各灰度值统计特性与图像灰度值的函数，它统计一幅图像中各个灰度级出现的次数或概率；从图形上来说，它是一个二维图，横坐标表示图像中各个像素点的灰度级，纵坐标为各个灰度级上图像各个像素点出现的次数或概率。

直方图均衡化是灰度变换的一个重要应用，广泛应用在图像增强处理中，它是以累计分布函数变换为基础的直方图修正法，可以产生一幅灰度级分布具有均匀概率密度的图像，扩展了像素的取值动态范围。

但是直方图均衡化存在着两个缺点：

(1) 变换后图像的灰度级减少，某些细节消失；

(2) 某些图像，如直方图有高峰，经处理后对比度不自然的过分增强。

为此 M.Kamel 和 Lian Guan 等人从图像相邻像素一般高度相关这一事实出发，将灰度概率分布和空间相关性联系在一起，提出了用二维条件概率密度函数取代一维概率密度函数作为均衡化条件，很好地解决了这个问题，有兴趣的读者可以参阅一些图像处理书籍和资料。

这里笔者将直方图均衡的源代码列出如下：

```
/*************************************************************************
*
* 函数名称：
*   Equalize()
*
* 参数：
*   HDIB hDIB
*
*
* 说明：
*   该函数用来对图像进行直方图均衡。
*
*************************************************************************/
```

```
void Equalize(HDIB hDIB)
{

BYTE* lpDIB=(BYTE*)::GlobalLock ((HGLOBAL)hDIB);

//获得像素数据区起始指针
BYTE* lpDIBBits=(BYTE*)::FindDIBBits((char*)lpDIB);

//获得图像宽度和高度

LONG lHeight=::DIBHeight ((char*)lpDIB);
LONG lWidth=::DIBWidth ((char*)lpDIB);

// 指向源图像的指针
unsigned char*  lpSrc;

// 临时变量
LONG    lTemp;

// 循环变量
LONG    i;
LONG    j;

// 灰度映射表
BYTE    bMap[256];

// 灰度映射表
LONG    lCount[256];

// 图像每行的字节数
LONG    lLineBytes;

// 计算图像每行的字节数
lLineBytes = WIDTHBYTES(lWidth * 8);

// 重置计数为 0
for (i = 0; i < 256; i ++)
{
    // 清零
    lCount[i] = 0;
}

// 计算各个灰度值的计数
for (i = 0; i < lHeight; i ++)
```

```
{
    for (j = 0; j < lWidth; j ++)
    {
        lpSrc = (unsigned char *)lpDIBBits + lLineBytes * i + j;

        // 计数加1
        lCount[*(lpSrc)]++;
    }
}

// 计算灰度映射表
for (i = 0; i < 256; i++)
{
    // 初始为0
    lTemp = 0;

    for (j = 0; j <= i ; j++)
    {
        lTemp += lCount[j];
    }

    // 计算对应的新灰度值
    bMap[i] = (BYTE) (lTemp * 255 / lHeight / lWidth);
}

// 每行
for(i = 0; i < lHeight; i++)
{
    // 每列
    for(j = 0; j < lWidth; j++)
    {
        // 指向DIB第i行，第j个像素的指针
        lpSrc= (unsigned char*)lpDIBBits + lLineBytes * (lHeight - 1 - i) + j;

        // 计算新的灰度值
        *lpSrc = bMap[*lpSrc];
    }
}

::GlobalUnlock ((HGLOBAL)hDIB);
}
```

添加菜单响应代码：

```
void CChildView::OnImgprcEqualize()
{
```

```
    //进行直方图均衡处理
    Equalize(m_hDIB);

    //显示处理结果
    CDC* pDC=GetDC();
    DisplayDIB(pDC,m_hDIB);
}
```

下面的两幅抓图是对"girl"图像做直方图均衡化前后的效果对比。

"girl"图像已经被作者加入了数字，可见，在直方图均衡化处理之前，数字几乎无法辨认；处理之后，则可以从女孩的背景中分辨出数字了。

原始图像如图 11-20 所示。直方图均衡之后效果如图 11-21 所示。



图 11-20　原始图像



图 11-21　直方图均衡化后效果图

可见，经过直方图均衡化，数字"0123"已经可以分辨了。

4. 用神经网络进行字符识别

（1）BP 神经网络简介

下面首先来简要介绍一下神经网络，然后再详细介绍 BP 网络，神经网络图如图 11-22 所示。

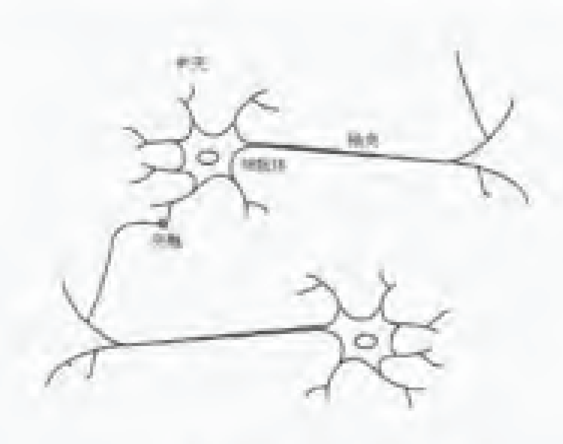


图 11-22　神经网络图

神经网络简介：人工神经网络（Artificial Neural Network,ANN），也称为神经网络（Neural Networks，NN），即从生物学神经系统的信号传递而抽象发展而成的一门学科。在神经网络

中，最基本的单元就是神经元。神经元由 3 部分组成：树突、细胞体和轴突。树突是树状的神经纤维接受网络，它将电信号传递给细胞体，细胞体对这些输入信号进行整合并进行阈值处理。轴突是单根长纤维，它把细胞体的输出信号导向其他的神经元。神经元的排列拓扑结构和突触的连接强度确立了神经网络的功能。形象的说，神经网络是由大量处理单元（神经元 Neurons）广泛连接而成的网络，是对人脑的抽象、简化和模拟，反映人脑的基本特性。它能够通过学习过程从外部环境中获取知识，并且它内部的很多的神经元可以用来存储这些已经学到的知识。

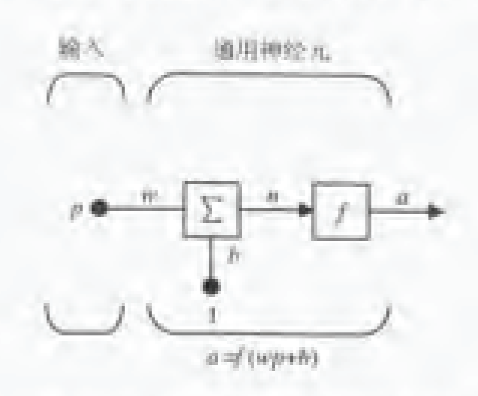


图 11-23　神经元抽象成数学模型

是将生物神经元模型抽象成一个信号传递的数学模型。

神经元的输入是信号 ***P***，经过一个累加器累加后的信号送入一个激活函数 *f*，从而得到这个神经元的输出 *a*。这个神经元的输出 *a* 同时又可以作为下一个或多个神经元的输入，从而将神经信号成网络分散状的传递出去。一个神经元可以接受多个输入，所以把神经元表示成为矢量、矩阵形式更容易去处理分析实际问题。

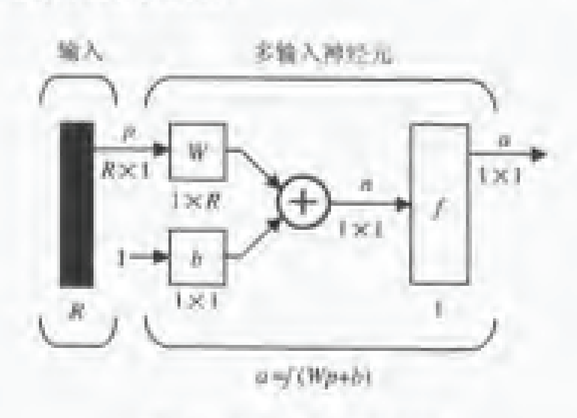


图 11-24　单维神经网络

图 11-24 中，***P*** 表示 *R* 维的输入向量，***B*** 为偏置向量，***W*** 为网络的权值矩阵，*f* 为激活传输函数，*a* 为网络的输出向量。

以上是对一个单层神经网络的描述。一般就实际情况而言，多层网络用处要广的多。后面用到的 BP 网络也是多层的。

在多层网络中，一般至少有 3 个层：一个输入层、一个输出层、一个或多个隐层。多层网络可以解决很多单层网络无法解决的问题，比如多层网络可以用来进行非线性分类，可以用来做精度极高的函数逼近，只要有足够多的层和足够多的神经元，这些都可以办到。

一个多层网络的输入和输出层的神经元个数是由外部描述定义的。例如如果有 4 个外部变量作为输入，那网络就要有 4 个输入。关于隐层神经元的确定，将在 BP 网络的设计中详细讨论。

文中曾多次提到过，感知器是神经网络的基础，也是 BP 网络的基础。所谓感知器（perceptron），也即给定一个或多个已知类别的输入，通过对网络的训练以实现所有输入数据的正确分类的数学模型。注意，感知器是单层网络。这里，训练的意思就是通过感知器的输出来反复调整网络的权值，以使其满足所有分类都正确的要求。不过一般来说，感知器的分类能力较差。

神经网络有好多种，比如径向基网络、BP 网络和 Hopfield 网络等。

本神经网络识别系统采用的是使用最为广泛的 BP 网络。

下面介绍 BP 学习算法，BP 学习过程可以描述如下。

工作信号正向传播：输入信号从输入层经过隐含层，传向输出层，在输出端产生输出信号，这是工作信号的正向传播。在信号传递的过程中网络的权值是固定不变的，每一层神经元的状态只影响下一层神经元的状态。如果在输出层得到的输出和期望输出的偏差比较大，则转入误差信号的反向传播。

误差信号反向传播：网络的实际输出和期望输出的差值就是误差信号。误差信号的反向传播就是误差信号从输出端传向输入端。在这个过程当中，网络的权值由误差反馈进行调节。通过不断的修改网络权值从而使得网络的输出不断的逼近期望值。

如图 11-25 所示是神经网络的示意图。如图 11-26 所示是多层 BP 网络结构示意图。

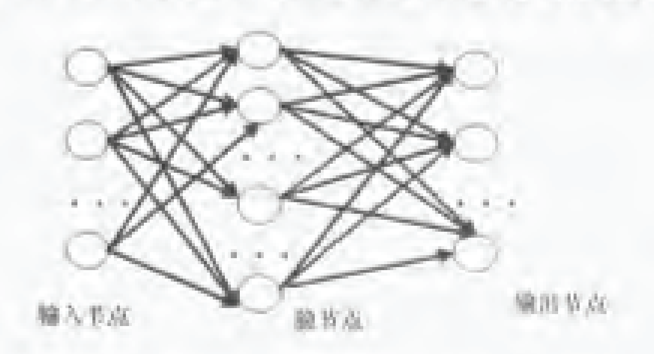


图 11-25　神经网络示意图

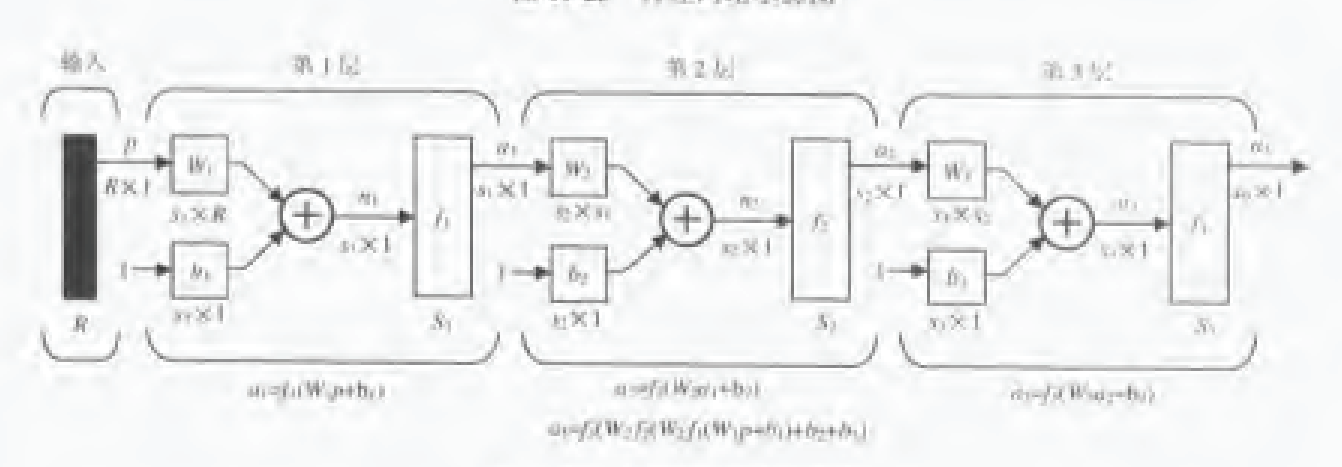


图 11-26　多层 BP 网络示意图

神经网络的激励函数。激励函数将输出信号压缩在一个允许的范围内，使其成为有限值，通常神经元输出的范围在[0,1]或者[-1,1]的闭区间上。常用的基本激励函数有阈值函数、分段线性函数、Sigmoid 函数。其中 Sigmoid 函数也称为 S 型函数，它是人工神经网络中用的最多的激励函数。S 型函数的定义如下：

$$f(x)=\frac{1}{1+e^{-x}} \tag{11-3}$$

神经网络的反馈调节。在误差信号的反向传播中，网络不断的修正各个节点的权值。

设有含 n 个节点的 BP 网络，采用 S 型激励函数。为简单起见，可以假设网络只有一个输出 y，任一节点的输出为 O_i，并设有 N 个样本(x_k,y_k)，$k=1,2,\cdots,N$)，对某一输入 x_k，网络的输出为 y_k，节点 i 的输出为 O_{ik}，节点 j 的输入为：

$$net_{jk}=\sum_i W_{ij}O_{ik} \tag{11-4}$$

使用平方型误差函数：

$$E_k=\frac{1}{2}\sum_{k=1}^{N}(y_k-\hat{y}_k)^2 \tag{11-5}$$

其中，y_k 为网络之实际输出，定义如下：

$$E_k=(y_k-\hat{y}_k)^2 \tag{11-6}$$

$$net_{jk}=\sum_i W_{ij}O_{ik} \tag{11-7}$$

$$\delta_{jk}=\frac{\partial E_k}{\partial net_{jk}} \tag{11-8}$$

其中，$O_{jk}=f(net_{jk})$ （11-9）

则：

$$\frac{\partial E_k}{\partial W_{ij}}=\frac{\partial E_k}{\partial net_{jk}}\frac{\partial net_{jk}}{\partial W_{ij}}=\frac{\partial E_k}{\partial net_{jk}}O_{ik} \tag{11-10}$$

当 j 为输出节点时，$Q_{jk}=\hat{y}_k$

$$\delta_{jk}=\frac{\partial E_k}{\partial \hat{y}_k}\frac{\partial \hat{y}_k}{\partial net_{jk}}=-(y_k-\hat{y}_k)f'(net_{jk}) \tag{11-11}$$

若 j 不是输出节点时，有：

$$\delta_{jk}=\frac{\partial E_k}{\partial net_{jk}}=\frac{\partial E_k}{\partial O_{jk}}\frac{\partial O_{jk}}{\partial net_{jk}}=\frac{\partial E_k}{\partial O_{jk}}\bullet f'(net_{jk}) \tag{11-12}$$

$$\begin{aligned}\frac{\partial E_k}{\partial O_{jk}} &= \sum_m \frac{\partial E_k}{\partial net_{mk}} \frac{\partial net_{mk}}{\partial O_{jk}} \\ &= \sum_m \frac{\partial E_k}{\partial net_{mk}} \frac{\partial}{\partial O_{jk}} \sum_i W_{mi} O_{ik} \\ &= \sum_m \frac{\partial E_k}{\partial net_{mk}} W_{mj} = \sum_m \delta_{mk} W_{mj}\end{aligned} \tag{11-13}$$

因此可得：

$$\begin{cases}\delta_{jk} = f'(net_{jk}) \sum\limits_m \delta_{mk} W_{mj} \\ \dfrac{\partial E_k}{\partial W_{ij}} = \delta_{jk} O_{jk}\end{cases}$$

BP 网络的学习过程。BP 学习算法可以归纳如下。

第 1 步，设置变量和参数，其中包括训练样本，权值矩阵，学习速率。

第 2 步，初始化，给各个权值矩阵一个较小的随机非零向量

第 3 步，输入随机样本。

第 4 步，对输入样本，前向计算 BP 网络每层神经元的输入信号和输出信号。

第 5 步，由实际输出和期望输出求得误差。判断是否满足要求，若满足转第 8 步；步满足转第 6 步。

第 6 步，判断是否已经到了最大迭代次数，若到，转第 8 步，否则反向计算每层神经元的局部梯度。

第 7 步，根据局部梯度修正各个矩阵的权值。

第 8 步，判断是否学习完所有的样本，是则结束，否则转第 3 步。

据笔者经验，BP 学习中需要注意的几点：

① 权值的初始化。权值的初始值应该选择均匀分布的小数经验值。初始值过大或者过小都会影响学习速度。为了避免权值的调整是同向的，应该将初始值设为随机数。

② 初始权值不要太大。否则可能会处于误差平面较平坦的区域，从而导致算法无法收敛，训练失败。

③ 神经元的激励函数是 s 型函数。所以如果函数的渐近值是 0，1 的话，期望输出只能是小于 1 大于 0 的数，而不能是 1 或者 0，否则可能会导致算法不收敛。在程序中建议读者用 0.1 来代表 0，0.9 代表 1。

（2）本程序中 BP 网络的设计及其编程实现

下面就开始用 BP 网络的思想来设计实现一个真正的神经网络。

BP 网络的一个重要的用途就是用于模式识别。任务是要设计并训练出一个可行、高效的 BP 网络，以实现对 0～9 共 10 个数字和识别。

经图像预处理过程之后，可以将最终提取到的字符的特征送入 BP 网络进行训练及识别了。这里，假设设定的字符标准归一化的宽度为 8，高度为 16，那么对于每个字符就有 128 维的特征。

设计 BP 网络的关键之处在于高效的特征提取方法、大量有代表性的训练样本、高效稳定

速收敛的学习方法。

BP 网络应用过程如图 11-27 所示。

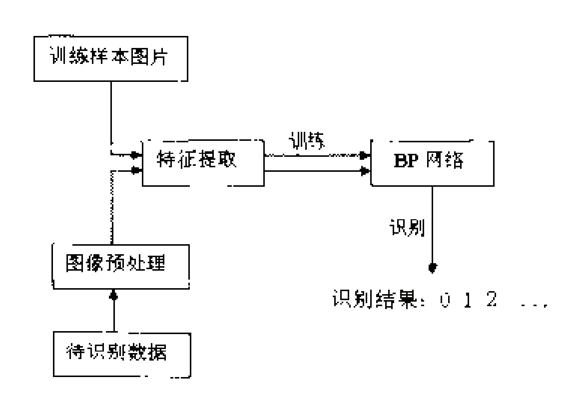


图 11-27　BP 网络应用流程

下面来讲讲 BP 网络 3 个层的神经元数目的确定。这也是 BP 网络设计的关键。

BP 网络应用的第一步就是要用已知训练样本来训练 BP 网络。这里，BP 网络的输入层的结点个数，为图像预处理后所输出的特征的维数。例如，采用了 8×16 归一化，特征提取采用的是逐个像素特征提取方法，也即直接利用每个点的像素值来作为特征，这样，对于每一个输入样本，就有 8×16＝128 个特征。所以就可以确定，输入层的结点数为 128。

对于隐层的结点数，没有什么硬性规定，一般来说，隐层神经元的数目越多，那么 BP 网络也就越精确，训练时间也越长。但要注意，隐层神经元不易选取太多，否则会造成识别率的急剧下降，也会降低网络的抗噪声能力。在本程序中，笔者推荐使用 10 个隐层神经元。读者可以自行测试一下，当将隐层神经元个数改为 30 个时，训练时间和识别率的变化。

对于输入层的结点数的确定，这决定于我们如何设定标准输出。也就是说如何对目标期望输出进行编码。在本程序中，笔者采用了 8421 码来对 0、1、2、3、4、5、6、7、8、9 来进行编码。对于输出“0”，采用（0,0,0,0）这样的目标输出向量来表示，对于输出“1”，采用（0,0,0,1）这样的输出向量来表示，同理，对于输出“9”，采用（1,0,0,1）这样的输出向量来表示。这样一来，就可以确定输出层的神经元数目为 4，也即为输出向量的维数。

其实通过后续编程实践可以发现，当采用（0,0,0,0）这样的目标输出向量的时候，BP 网络无法收敛。那是因为采用的激活函数（传输函数）的输出永远不可能达到 0 或 1，而只能是接近。所以，在这里要纠正一下目标输出向量。对其重新编码后，最终确定编码方案如下：

```
0 的编码：      0.1,0.1,0.1,0.1
1 的编码：      0.1,0.1,0.1,0.9
2 的编码：      0.1,0.1,0.9,0.1
3 的编码：      0.1,0.1,0.9,0.9
4 的编码：      0.1,0.9,0.1,0.1
5 的编码：      0.1,0.9,0.1,0.9
6 的编码：      0.1,0.9,0.9,0.1
7 的编码：      0.1,0.9,0.9,0.9
8 的编码：      0.9,0.1,0.1,0.1
9 的编码：      0.9,0.1,0.1,0.9
```

使用 BP 网络来进行数字识别的流程如下。

首先，利用大量的训练样本来训练网络，以得到以文件形式保存的权值。训练样本为精心选择的可以很好的反应样本可分性特性的已知数据。在程序中采用训练样本图片的形式。将训练样本图片进行特征提取后，就可以送入 BP 网络进行训练。在这里，采用了含有 40 个字符数据的图像作为训练样本。这幅图像包含了 Arial 字体书写的普通的 0 到 9 的 10 个数字、斜体的 10 个数字、粗体的 10 个数字和 10 个倾斜的数字，总共 40 个，如图 11-28 所示。

0123456789

0123456789

0123456789

0123456789

图 11-28　40 个训练样本

实验证明，这 40 个训练样本训练出来的 BP 网络对于 Arial 字体的数字可以达到 90%以上的识别率，而且网络训练时间也是可以接受的（大约 1～5s）。当然如果采用 400 个训练样本甚至更多的话，那无疑可以进一步提高识别率，但训练时间往往会达到分钟甚至更高到小时的级别，实际意义不大了。

其次，训练完 BP 网络后，就可以用它对为止数据进行识别了。识别首先要经过图像预处理、特征提取，最后送入 BP 网络识别，直接得到结果。

在训练之前，程序要求输入训练参数，如训练误差、步长等，如图 11-29 所示。

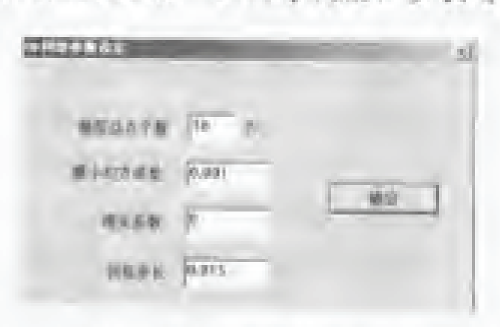


图 11-29　BP 训练参数输入框

下面作者列出 BP 网络训练及识别的完整源代码 bp.h。

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>
#define BIGRND 32767

/*************************函数声明部分*****************************/
```

```
//随机数产生函数
double drnd();
double dpn1();

//S函数
double squash(double x);

//分配1维double型的内存
double *alloc_1d_dbl(int n);

//分配2维double型的内存
double **alloc_2d_dbl(int m, int n);

//初始化BP网络
void bpnn_initialize(int seed);

//随机初始化权值
void bpnn_randomize_weights(double **w, int m, int n);

//零初始化权值
void bpnn_zero_weights(double **w, int m, int n);

//信息的前向传输
void bpnn_layerforward(double *l1, double *l2, double **conn, int n1, int n2);

//误差输出
void bpnn_output_error(double *delta, double *target, double *output, int nj);

//隐层误差
void bpnn_hidden_error(double* delta_h, int nh, double *delta_o, int no, double **who, double *hidden);

//根据误差调整权值
void bpnn_adjust_weights(double *delta, int ndelta, double *ly, int nly, double** w, double **oldw, double eta, double momentum);

//保存权值
void w_weight(double **w,int n1,int n2,char*name);

//读取权值
bool r_weight(double **w,int n1,int n2,char *name);

//保存Bp网络各层结点数目
void w_num(int n1,int n2,int n3,char*name);
```

```
//读取 Bp 网络各层结点数目
bool r_num(int *n,char *name);

//特征提取
void code(BYTE*image ,int *p,int w,int h,int dw);

//BP 网络训练
void BpTrain(HDIB hDIB,int n_hidden,double min_ex,double momentum,double
eta ,int width,int height);

//利用 BP 网络进行识别
void CodeRecognize(HDIB hDIB,int width ,int height ,int n_in ,int n_hidden,int
n_out);
/*****************************************************************************/
/****************** 以下是函数的实现部分*************************************/

/*** 返回 0—1 的双精度随机数 ***/
double drnd()
{
//BIGRND 定义为随机数的最大范围
return ((double) rand() / (double) BIGRND);
}

/*** 返回-1.0 到 1.0 之间的双精度随机数 ***/
double dpn1()
{
return ((drnd() * 2.0) - 1.0);
}

double squash(double x)
{
//返回 S 激活函数
return (1.0 / (1.0 + exp(-x)));
}

/*** 申请 1 维双精度实数数组 ***/
double *alloc_1d_dbl(int n)
{
double *new1;

//申请内存
new1 = (double *) malloc ((unsigned) (n * sizeof (double)));
```

```
//申请内存失败处理
if (new1 == NULL) {
    printf("ALLOC_1D_DBL: Couldn't allocate array of doubles\n");
    return (NULL);
}

//返回申请到的内存的指针
return (new1);
}

/*** 申请 2 维双精度实数数组 ***/
double **alloc_2d_dbl(int m, int n)
{
int i;

//定义一二维指针
double **new1;

//先申请一维内存
new1 = (double **) malloc ((unsigned) (m * sizeof (double *)));

//申请失败处理
if (new1 == NULL) {
    //  printf("ALLOC_2D_DBL: Couldn't allocate array of dbl ptrs\n");
    return (NULL);
}

//再申请二维内存。一维内存中存放的是指针
for (i = 0; i < m; i++) {
    new1[i] = alloc_1d_dbl(n);
}

//返回申请到的二维内存(可以看作矩阵)
return (new1);
}


/*** 设置随机数种子 ***/
void bpnn_initialize(int seed)
{
//printf("Random number generator seed: %d\n", seed);
srand(seed);
}
```

```
/*** 随机初始化权值 ***/
void bpnn_randomize_weights(double **w, int m, int n)
{
 int i, j;
 //调用 dpn1 随机初始化权值
 for (i = 0; i <= m; i++) {
     for (j = 0; j <= n; j++) {

         w[i][j] = dpn1();
     }
 }
}

/******* 零初始化权值 *******/
void bpnn_zero_weights(double **w, int m, int n)
{
 int i, j;

 //将权值逐个赋 0
 for (i = 0; i <= m; i++) {
     for (j = 0; j <= n; j++) {
         w[i][j] = 0.0;
     }
 }
}

/*********前向传输*********/
void bpnn_layerforward(double *l1, double *l2, double **conn, int n1, int n2)
{
 double sum;
 int j, k;

 /*** 设置阈值 ***/
 l1[0] = 1.0;

 /*** 对于第二层的每个神经元 ***/
 for (j = 1; j <= n2; j++) {

     /*** 计算输入的加权总和 ***/
     sum = 0.0;
     for (k = 0; k <= n1; k++) {
         sum += conn[k][j] * l1[k];
     }
```

```
        l2[j] = squash(sum);
    }

}

/* 输出误差 */
void bpnn_output_error(double *delta, double *target, double *output, int nj)
{
 int j;
 double o, t, errsum;

 //先将误差归零
 errsum = 0.0;

 //循环计算 delta
 for (j = 1; j <= nj; j++) {
     o = output[j];
     t = target[j];
     //计算 delta 值
     delta[j] = o * (1.0 - o) * (t - o);

 }

}

/* 隐含层误差 */

void bpnn_hidden_error(double* delta_h, int nh, double *delta_o, int no, double
**who, double *hidden)
{
 int j, k;
 double h, sum, errsum;

 //误差归零
 errsum = 0.0;

 //计算新 delta
 for (j = 1; j <= nh; j++) {
     h = hidden[j];
     sum = 0.0;
     for (k = 1; k <= no; k++) {
         sum += delta_o[k] * who[j][k];
     }
```

```
        delta_h[j] = h * (1.0 - h) * sum;

    }

    }


    /* 调整权值 */
    void bpnn_adjust_weights(double *delta, int ndelta, double *ly, int nly,
double** w, double **oldw, double eta, double momentum)
    {
     double new_dw;
     int k, j;

     ly[0] = 1.0;
     //请参考文章中 BP 网络权值调整的计算公式
     for (j = 1; j <= ndelta; j++) {
         for (k = 0; k <= nly; k++) {
             new_dw = ((eta * delta[j] * ly[k]) + (momentum * oldw[k][j]));
             w[k][j] += new_dw;
             oldw[k][j] = new_dw;
         }
     }
    }

    /*******保存权值**********/
    void w_weight(double **w,int n1,int n2,char*name)
    {
     int i,j;
     double *buffer;

     //创建文件指针
     FILE *fp;

     //打开文件
     fp=fopen(name,"wb+");

     //分配缓冲区
     buffer=(double*)malloc((n1+1)*(n2+1)*sizeof(double));

     //填写缓冲区内容
     for(i=0;i<=n1;i++)
     {
         for(j=0;j<=n2;j++)
```

```
        buffer[i*(n2+1)+j]=w[i][j];
}

//将缓冲区内容写入文件
fwrite((char*)buffer,sizeof(double),(n1+1)*(n2+1),fp);

//关闭文件
fclose(fp);

//清空缓冲区
free(buffer);
}


/*************读取权值*************/
bool  r_weight(double **w,int n1,int n2,char *name)
{
 int i,j;

 //临时缓冲区指针
 double *buffer;

 //文件指针
 FILE *fp;

 //判断是否可以正确打开文件。若失败则返回 false
 if((fp=fopen(name,"rb"))==NULL)
 {
     ::MessageBox(NULL,"无法读取权值信息",NULL,MB_ICONSTOP);
     return (false);
 }

 //将临时缓冲区指针执行新申请的内存
 buffer=(double*)malloc((n1+1)*(n2+1)*sizeof(double));

 //读取文件内容到缓冲区
 fread((char*)buffer,sizeof(double),(n1+1)*(n2+1),fp);

 //由缓冲区内容填写权值
 for(i=0;i<=n1;i++)
 {
     for(j=0;j<=n2;j++)
         w[i][j]=buffer[i*(n2+1)+j];
 }
```

```
 //关闭文件
 fclose(fp);

 //释放临时缓冲区
 free(buffer);

 //返回true表示已经正确读取
 return(true);
}


/*****保存Bp网络各层结点的数目******/
void w_num(int n1,int n2,int n3,char*name)
{
 //文件指针
 FILE *fp;

 //打开文件
 fp=fopen(name,"wb+");

 //缓冲区指针
 int *buffer;
 //分配内存并指向缓冲区指针
 buffer=(int*)malloc(3*sizeof(int));

 //将网络各层参数信息保存在临时缓冲区
 buffer[0]=n1;

 buffer[1]=n2;

 buffer[2]=n3;

 //将临时缓冲区内容写入文件
 fwrite((char*)buffer,sizeof(int),3,fp);

 //关闭文件
 fclose(fp);

 //清空缓冲区
 free(buffer);

}
```

```
/********读取 Bp 网络各层结点数目*********/
bool r_num(int *n,char *name)
{
 //临时缓冲区指针
 int *buffer;

 //文件指针
 FILE *fp;

 //分配内存空间
 buffer=(int *)malloc(3*sizeof(int));

 if((fp=fopen(name,"rb"))==NULL)
 {
     ::MessageBox(NULL,"结点参数",NULL,MB_ICONSTOP);
     return (false);
 }

 //读取文件到缓冲区
 fread((char*)buffer,sizeof(int),3,fp);

 //从缓冲区中取出数据，存入 n[0]~n[2]
 n[0]=buffer[0];
 n[1]=buffer[1];
 n[2]=buffer[2];

 //关闭文件
 fclose(fp);

 //清空缓冲区
 free(buffer);

 //正确读取，返回 true
 return(true);
}

/**********************************
 * 函数名称 code()
 *
 * 参量:
 *   BYTE* lpDIBBits    -指向输入图像的像素其实位置的指针
 *   int num            -图片中样本的个数
 *   LONG lLineByte     -输入图片每行的字节数
 *   LONG lSwidth       -预处理时归一化的宽度
```

```
 *   LONG lSheight      -预处理时归一化的长度
 *
 *  函数功能 :
 *       对于输入样本提取特征向量，在这里把归一化样本的
 *        每一个像素都作为特征提取出来
 ***************************************/

double** code (BYTE* lpDIBBits,int num, LONG lLineByte,LONG lSwidth,LONG
lSheight)
 {
  //循环变量
  int i,j,k;
  BYTE* lpSrc;

  // 建立保存特征向量的二维数组
  double **data;

  // 为这个数组申请二维存储空间
  data = alloc_2d_dbl(num,lSwidth*lSheight);

  // 将归一化的样本的每个像素作为一个特征点提取出来

  //逐个数据扫描
  for(k=0;k<num;k++)
  {
      //对每个数据逐行扫描
      for(i=0;i<lSheight;i++)
      {
          //对每个数据逐列扫描
          for(j=k*lSwidth;j<(k+1)*lSwidth;j++)
          {

              // 指向图像第 i 行第 j 列个像素的指针
              lpSrc = lpDIBBits + i*lLineByte + j;

              //如果这个像素是黑色的
              if(*(lpSrc)==0)
                  //将特征向量的相应位置填 1
                  data[k][i*lSwidth+j-k*lSwidth]=1;

              //如果这个像素是其他的
              if(*(lpSrc)!=0)
                  //将特征向量的相应位置填 0
                  data[k][i*lSwidth+j-k*lSwidth]=0;
```

```
            }
        }
    }
    //返回指向特征矩阵的二维指针
    //特征矩阵构成：每个样本×每个样本的特征
    return(data);
}

/*******************************************************
* 函数名称 BpTrain()
*
* 参数:
*   double **data_in      -指向输入的特征向量数组的指针
*    double **data_out    -指向理想输出数组的指针
int n_in                  -输入层结点的个数
*   int n_hidden          -BP 网络隐层结点的数目
*   double min_ex         -训练时允许的最大均方误差
*   double momentum       -BP 网络的相关系数
*   double eta            -BP 网络的训练步长
*   int num               -输入样本的个数
*
* 函数功能:
*      根据输入的特征向量和期望的理想输出对 BP 网络尽行训练
*      训练结束后将权值保存并将训练的结果显示出来
*******************************************************/
void BpTrain(double ** data_in, double** data_out,int n_in,int n_hidden,double
min_ex,double momentum,double eta ,int num)
{


    //循环变量
    int i,k,l;

    //输出层结点数目
    int  n_out=4;

    //指向输入层数据的指针
    double* input_unites;

    //指向隐层数据的指针
    double* hidden_unites;
```

```
//指向输出层数据的指针
double* output_unites;

//指向隐层误差数据的指针
double* hidden_deltas;

//指向输出层误差数剧的指针
double* output_deltas;

//指向理想目标输出的指针
double* target;

//指向输入层于隐层之间权值的指针
double** input_weights;

//指向隐层与输出层之间的权值的指针
double** hidden_weights;

//指向上一此输入层于隐层之间权值的指针
double** input_prev_weights ;

//指向上一此隐层与输出层之间的权值的指针
double** hidden_prev_weights;

//每次循环后的均方误差误差值
double ex;

//为各个数据结构申请内存空间
input_unites= alloc_1d_dbl(n_in + 1);
hidden_unites=alloc_1d_dbl(n_hidden + 1);
output_unites=alloc_1d_dbl(n_out + 1);
hidden_deltas = alloc_1d_dbl(n_hidden + 1);
output_deltas = alloc_1d_dbl(n_out + 1);
target = alloc_1d_dbl(n_out + 1);
input_weights=alloc_2d_dbl(n_in + 1, n_hidden + 1);
input_prev_weights = alloc_2d_dbl(n_in + 1, n_hidden + 1);
hidden_prev_weights = alloc_2d_dbl(n_hidden + 1, n_out + 1);
hidden_weights = alloc_2d_dbl(n_hidden + 1, n_out + 1);

//为产生随机序列撒种
time_t t;
bpnn_initialize((unsigned)time(&t));

//对各种权值进行初始化初始化
```

```
bpnn_randomize_weights( input_weights,n_in,n_hidden);
bpnn_randomize_weights( hidden_weights,n_hidden,n_out);
bpnn_zero_weights(input_prev_weights, n_in,n_hidden );
bpnn_zero_weights(hidden_prev_weights,n_hidden,n_out );

//开始进行 BP 网络训练
//这里设定最大的迭代次数为 15000 次
for(l=0;l<15000;l++)
{
    //对均方误差置零
    ex=0;
    //对样本进行逐个的扫描
    for(k=0;k<num;k++)
    {
        //将提取的样本的特征向量输送到输入层上
        for(i=1;i<=n_in;i++)
            input_unites[i] = data_in[k][i-1];

        //将预定的理想输出输送到 BP 网络的理想输出单元
        for(i=1;i<=n_out;i++)
            target[i]=data_out[k][i-1];

        //前向传输激活

        //将数据由输入层传到隐层
        bpnn_layerforward(input_unites,hidden_unites,
            input_weights, n_in,n_hidden);

        //将隐层的输出传到输出层
        bpnn_layerforward(hidden_unites, output_unites,
            hidden_weights,n_hidden,n_out);

        //误差计算

        //将输出层的输出与理想输出比较计算输出层每个结点上的误差
        bpnn_output_error(output_deltas,target,output_unites,n_out);

        //根据输出层结点上的误差计算隐层每个节点上的误差
        bpnn_hidden_error ( hidden_deltas, n_hidden, output_deltas, n_out,
hidden_weights, hidden_unites);

        //权值调整
        //根据输出层每个节点上的误差来调整隐层与输出层之间的权值
        bpnn_adjust_weights(output_deltas,n_out, hidden_unites,n_hidden,
```

```
            hidden_weights, hidden_prev_weights, eta, momentum);

        //根据隐层每个节点上的误差来调整隐层与输入层之间的权值
        bpnn_adjust_weights(hidden_deltas, n_hidden, input_unites, n_in,
            input_weights, input_prev_weights, eta, momentum);

        //误差统计
        for(i=1;i<=n_out;i++)

ex+=(output_unites[i]-data_out[k][i-1])*(output_unites[i]-data_out[k][i-1]);
      }

      //计算均方误差
      ex=ex/double(num*n_out);

      //如果均方误差已经足够的小，跳出循环，训练完毕
      if(ex<min_ex)break;
   }

   //相关保存

   //保存输入层与隐层之间的权值
   w_weight(input_weights,n_in,n_hidden,"win.dat");

   //保存隐层与输出层之间的权值
   w_weight(hidden_weights,n_hidden,n_out,"whi.dat");

   //保存各层结点的个数
   w_num(n_in,n_hidden,n_out,"num");

   //显示训练结果

   CString str;
   if(ex<=min_ex)
   {
       str.Format ("迭代%d 次，\n 平均误差%.4f",l,ex);

       ::MessageBox(NULL,str,"训练结果",NULL);
   }

   if(ex>min_ex)
   {

       str.Format("迭代%d 次，平均误差%.4f\n 我已经尽了最大努力了还是达不到您的要求\n 请调
```

```
整参数重新训练吧！",1,ex);
        ::MessageBox(NULL,str,"训练结果",NULL);
    }

    //释放内存空间

    free(input_unites);
    free(hidden_unites);
    free(output_unites);
    free(hidden_deltas);
    free(output_deltas);
    free(target);
    free(input_weights);
    free(hidden_weights);
    free(input_prev_weights);
    free(hidden_prev_weights);
    }

    /*********************************************
    * 函数名称
    * CodeRecognize()
    * 参量
    *  double **data_in      -指向待识别样本特征向量的指针
    *  int num              -待识别的样本的个数
    *  int n_in             -Bp 网络输入层结点的个数
    *  int n_hidden         -Bp 网络隐层结点的个数
    *  int n_out            -Bp 网络输出层结点的个数
    * 函数功能:
    *    读入输入样本的特征相量并根据训练所得的权值
    *    进行识别，将识别的结果写入 result.txt
    ******************************************/

    void CodeRecognize(double **data_in, int num ,int n_in,int n_hidden,int n_out)
    {
     //循环变量
     int i,k;
     // 指向识别结果的指针
     int *recognize;
     //为存放识别的结果申请存储空间
     recognize=(int*)malloc(num*sizeof(int));

     //指向输入层数据的指针
     double* input_unites;
```

```
//指向隐层数据的指针
double* hidden_unites;

//指向输出层数据的指针
double* output_unites;

//指向输入层于隐层之间权值的指针
double** input_weights;

//指向隐层与输出层之间的权值的指针
double** hidden_weights;

//为各个数据结构申请内存空间
input_unites= alloc_1d_dbl(n_in + 1);
hidden_unites=alloc_1d_dbl(n_hidden + 1);
output_unites=alloc_1d_dbl(n_out + 1);
input_weights=alloc_2d_dbl(n_in + 1, n_hidden + 1);
hidden_weights = alloc_2d_dbl(n_hidden + 1, n_out + 1);


//读取权值
if( r_weight(input_weights,n_in,n_hidden,"win.dat")==false)
    return;
if(r_weight(hidden_weights,n_hidden,n_out,"whi.dat")==false)
    return;


//逐个样本扫描
for(k=0;k<num;k++)
{
    //将提取的样本的特征向量输送到输入层上
    for(i=1;i<=n_in;i++)
        input_unites[i]=data_in[k][i-1];

    //前向输入激活
    bpnn_layerforward(input_unites,hidden_unites,
        input_weights, n_in,n_hidden);

    bpnn_layerforward(hidden_unites, output_unites,
        hidden_weights,n_hidden,n_out);

    //根据输出结果进行识别
    int result=0 ;
    //考察每一位的输出
```

```
    for(i=1;i<=n_out;i++)
    {
        //如果大于 0.5 判为 1
        if(output_unites[i]>0.5)

            result+=(int)pow(2,double(4-i));
    }

    //如果判定的结果小于等于 9，认为合理
    if(result<=9)
        recognize[k]=result;
    //如果判定的结果大于 9，认为不合理将结果定位为一个特殊值 20
    if(result>9)
        recognize[k]=20;
}


//将识别结果写到文本中
FILE *fp;
fp=fopen("result.txt","w+");

for(i=0;i<num;i++)
{
    if(recognize[i]==20)
        fprintf(fp,"无法识别，");
    else
        fprintf(fp,"%d,",recognize[i]);

}
fclose(fp);

//将识别的结果显示出来
CString str,str1;
for(i=0;i<num;i++)
{
    if(recognize[i]!=20)
        str.Format("%d ",recognize[i]);

    if(recognize[i]==20)
        str.Format("无法识别 ");

    str1+=str;
}
```

```
//通知用户训练完成

::MessageBox(NULL,str1,"识别结果",NULL);

//释放存储空间

free(input_unites);
free(hidden_unites);
free(output_unites);
free(input_weights);
free(hidden_weights);
}
```

下面是菜单事件响应函数。

训练 BP 网络：

```
void CChildView::OnBpnetTrain()
{
 OnImgprcAll();
 //判断是否经过了归一划的处理
 if(gyhfinished==false)
 {
     //如果没有进行提示错误并返回
     ::MessageBox(NULL,"没有进行归一划预处理",NULL,MB_ICONSTOP);
     return;
 }

 //建立 BP 网络训练参数对话框

 CDBpParamater BpPa;

 //用默认值初始化变量
 //动量因子
 BpPa.m_a=0;
 //学习步长
 BpPa.m_eta=0.015;
 //允许误差
 BpPa.m_ex=0.001;
 //隐层神经元数目
 BpPa.m_hn=10;

 // 显示对话框
 if(BpPa.DoModal()!=IDOK)
 {
     //返回
     return;
```

```
}
//用户获得参数信息

//相关系数(动量因子)
double  momentum=BpPa.m_a;

//最小均方误差
double  min_ex=BpPa.m_ex;

//隐层数目
int  n_hidden=BpPa.m_hn;

//训练步长
double eta=BpPa.m_eta;

//获得指向DIB的指针
BYTE *lpDIB=(BYTE*)::GlobalLock((HGLOBAL) m_hDIB);

//获得指向DIB像素的指针，并指向像素的起始位置
BYTE *lpDIBBits =(BYTE*)::FindDIBBits((char *)lpDIB);

//获得颜色信息
int numColors=(int) ::DIBNumColors((char *)lpDIB);

//不是灰度图返回
if (numColors!=256)
{
    ::GlobalUnlock((HGLOBAL) m_hDIB);
    ::MessageBox(NULL,"只能处理灰度图像",NULL,MB_ICONSTOP);
    return;
}

//获取图像的宽度
LONG lWidth = (LONG) ::DIBWidth((char *)lpDIB);

//获取图像的高度
LONG lHeight = (LONG) ::DIBHeight((char *)lpDIB);

//计算图像每行的字节数
LONG lLineByte = (lWidth+3)/4*4;

//归一化的宽度
LONG lSwidth = w_sample;
```

```
//归一化的高度
LONG LSheight = h_sample;

//指向输入样本的特征向量的指针
double **data_in;

//从输入的训练样本中提取特征向量
data_in = code ( lpDIBBits, digicount,  lLineByte, lSwidth, LSheight);

//计算输入层结点的数目

int n_in = LSheight*lSwidth;

//设定目标输出向量
double out[][4]=
{      0.1,0.1,0.1,0.1,
0.1,0.1,0.1,0.9,
0.1,0.1,0.9,0.1,
0.1,0.1,0.9,0.9,
0.1,0.9,0.1,0.1,
0.1,0.9,0.1,0.9,
0.1,0.9,0.9,0.1,
0.1,0.9,0.9,0.9,
0.9,0.1,0.1,0.1,
0.9,0.1,0.1,0.9};


//指向输出
double **data_out;

data_out = alloc_2d_dbl(digicount,4);

for(int i=0;i<digicount;i++)
{
    for(int j=0;j<4;j++)
        data_out[i][j]=out[i%10][j];

}

//训练 BP 网络
BpTrain(data_in,data_out,in,n_hidden,min_ex,momentum,eta,digicount);

//解除对位图的锁定
::GlobalUnlock(n_hDIB);
```

```
//屏幕显示
CDC* pDC=GetDC();
DisplayDIB(pDC,m_hDIB);
}
```

利用训练好的 BP 网络去识别未知数据：

```
void CChildView::OnBpnetRecognize()
{
// 先进行一次性图像预处理
OnImgprcAll();

//判断是否经过了归一划的处理
if(gyhfinished==false)
{
    //如果没有进行提示错误并返回
    ::MessageBox(NULL,"没有进行归一划预处理",NULL,MB_ICONSTOP);
    return;
}

//获得指向 DIB 的指针
BYTE *lpDIB=(BYTE*)::GlobalLock((HGLOBAL) m_hDIB);

//获得指向 DIB 像素的指针，并指向像素的起始位置
BYTE *lpDIBBits =(BYTE*)::FindDIBBits((char *)lpDIB);

//获得颜色信息
int numColors=(int) ::DIBNumColors((char *)lpDIB);

//不是灰度图返回
if (numColors!=256)
{
    ::GlobalUnlock((HGLOBAL) m_hDIB);
    ::MessageBox(NULL,"只能处理 256 色图像",NULL,MB_ICONSTOP);
    return;
}

//获取图像的宽度
LONG lWidth = (LONG) ::DIBWidth((char *)lpDIB);

//获取图像的高度
LONG lHeight = (LONG) ::DIBHeight((char *)lpDIB);

//计算图像每行的字节数
LONG lLineByte = (lWidth+3)/4*4;
```

```
//归一化的宽度
LONG lSwidth = w_sample;

//归一化的高度
LONG LSheight = h_sample;

// 读取结点信息
int n[3];
if(r_num(n,"num")==false)
    return;

//获得输入层结点数目
int  n_in=n[0];

//获得隐层结点数目
int  n_hidden=n[1];

//获得输出层结点数目
int  n_out=n[2];

//判断待识别样本的归一划信息是否与训练时相同
if(n_in!=lSwidth*LSheight)
{
    //如果不相同提示错误并返回
    ::MessageBox(NULL,"归一划尺寸与上一次训练时不一致",NULL,MB_ICONSTOP);
    return;
}

//指向输入样本的特征向量的指针
double **data_in;
//从输入的训练样本中提取特征向量
data_in = code ( lpDIBBits, digicount,  lLineByte, lSwidth, LSheight);

//根据提取的特征进行样本识别
CodeRecognize(data_in, digicount,n_in,n_hidden,n_out);

//解除对位图的锁定
::GlobalUnlock(m_hDIB);

//显示位图
CDC* pDC=GetDC();
DisplayDIB(pDC,m_hDIB);
}
```

利用该 BP 网络对训练样本进行训练的结果如图 11-30 所示。最终对图 11-4 的识别结果如图 11-31 所示。





图 11-30　训练结果



图 11-31　识别结果

同时识别结果还将输出与 result.txt 文件中进行保存，如图 11-32 所示。

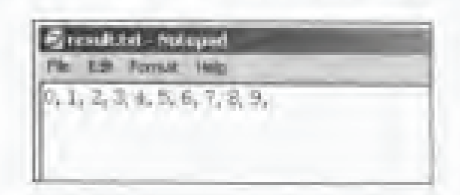


图 11-32　识别结果保存

（3）矩阵形式的 BP 网络

来看一下矩阵形式表示的 BP 网络，在图 11-18 所描述的网络中，算法的输入是一系列正确的、已知的样本和输出对：{$P1,T1$}，{$P2,T2$}，{$P3,T3$}……

网络输出误差为：

$$F(x)=E(e^{T}e)$$

BP 网络通过微积分中的链法则来计算偏导数。这里给出训练 BP 网络的公式，详细的推导过程请见相关参考文献。

首先是信息的正向传播：

$$a^0 = P$$

$$a^{m+1}=f^{m+1}(w^{m+1}a^m+b^{m+1}),\ m=0,1,\ldots,M-1$$

$$a = a^M$$

这里 P 为输入样本向量，a为网络各层的输出向量，f 为激活函数（传输函数）的矩阵形式，W 为各层的权值矩阵，b 为偏置。

然后是误差（这里称为敏感性）的反向传播：

$$a^M=-2F^M(n^M)\cdot(t-a)$$

$$s^m=F^m(n^m)\cdot(W^{m+1})^T\cdot s^{m+1},\ m=M-1,\ldots,2,1$$

式中，s 为敏感性。

最后是使用最速下降法来更新权值和偏置：

$$W^m(k+1)=W^m(k)-\alpha \cdot s^m \cdot (a^{m-1})T$$

$$b^m(k+1)=b^m(k)-\alpha \cdot s^m$$

矩阵形式的BP网络形式简单。请读者参阅《神经网络设计》中有详述。这里限于篇幅，仅给读者提供一个大体框架。

可见，矩阵形式表示的 BP 网络结构很清晰明了。作者编程的时候没有采用矩阵形式的BP网络，那是基于速度的考虑。如果基于编程实现的简单性考虑的话，那么读者可以尝试使用一下矩阵形式的BP网络。

笔者在随书光盘中提供了一个简单的矩阵运算类库tmat.h和向量运算类库tvec.h，内含简要的调用说明。读者可以自行试一下利用矩阵类库去实现BP网络，会有意想不到的编程实现上的简单。但是速度就会有明显的下降了。

比如，要实现信息正向传播中的：$N=W\times A+B$，就可以这么写：

```
//首先要加入类库的头文件。这里包括两个：tvec.h和tmat.h
#include "tvec.h"
#include "tmat.h"
#include <crtdbg.h>
typedef tmat<double> CMatrix;
typedef tvec<double> CVector;
//此时就可以用CMatrix和CVector来定义矩阵和向量对象了
//矩阵对象可以是权值，偏置，而输入和标准输出则可以用向量来表示
CMatrix W,A,B,N;
N=W*A+B;
//就这么简单！当然，还有很多前叙工作这里没做。这里只是简单说明一下
```

矩阵类库的完整代码我在这里就不列出了，仅仅举一个用类来实现重载“+”运算符的简单的例子：

```
template<class T>
tmat<T> operator + (tmat<T> a, tmat<T> b)
{
 //判断两个矩阵的行、列是否相等，否则不予完成相加操作
 if (a.nrows() != b.nrows() || a.ncols() != b.ncols()) {
     a.cerror("matrices do not conform in addition\n");
     return a;
 }

 //新建立一个临时矩阵，存放相加的结果
 tmat<T> c(a.nrows(), a.ncols());
 //执行矩阵元素逐个相加的运算
 for (long i = 1; i <= c.nrows(); i++)
     for (long j = 1; j <= c.ncols(); j++)
         c(i, j) = a(i, j) + b(i, j);
     //返回相加后的矩阵
```

```
    return c;
}
```

可见该矩阵类库采用的是值传递。采用值传递，函数调用比较简单，也不必考虑内存释放的问题，但却带来了效率的降低。

下面给出调用该矩阵类库的方法和几个简单实例，以供读者参考。

源代码位于 test.cpp 中；矩阵和向量库为 tmat.h 和 tvec.h。

```
//test.cpp    矩阵的测试
#include "tvec.h"
#include "tmat.h"
#include <crtdbg.h>
typedef tmat<double> CMatrix;
typedef tvec<double> CVector;


/*************************************************************************
一个简单易用的矩阵类库
使用简介：

  初始化矩阵： tmax<double>A(3,2);          -生成一个 3 行 2 列的零阵
  初始化矩阵： tmax<double>B(2,2,a);        -生成一个 2 行 2 列的矩阵并利用数组 a 的前 2×2
个元素为其赋值。这里要求 a 至少要包含 2×2 个元素
  给矩阵元素赋值： A(1,2)=1.0;           -将 1.0 赋给矩阵 A 的第 1 行、第 2 列
  取矩阵元素的值： f=A(2,2);               -取出矩阵 A 第 2 行、第 2 列的值赋给 f
  矩阵的加、减、乘： A=B+C;C=B*A;等等，可以像普通数据类型那样直接进行运算符运算
  矩阵转置：      transpose(A);
  求矩阵行列式：   det(A);     注意：必须是方阵
  矩阵求逆：      inv(A);      注意：必须是方阵

 得到矩阵的行数和列数：  A.nrows();   A.ncols();

  resize 矩阵：  A.resize(2,2);           -取原来 A 矩阵的前 2×2 个元素来构造新尺寸的 A


 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

      另外，tvec.h 则包含了一个矢量(向量)类库，这里也简要介绍一下

          初始化向量：  tvec<double>A;            -生成一个一维的向量。默认初始值为 0
          初始化向量：  tvec<double>B(4);         -生成一个 4 维的向量
          初始化向量：  tvec<double>C(4,0.1);   -生成一个 4 维的向量，并将每一维元素初始
化为 0.1
          初始化向量：  tvec<double>D(5,a);   -生成一个 5 维的向量，并用数组 a 的前 5 个元
素为其赋值
          初始化向量：  tvec<double>E(B);         -由向量 B 来初始化向量 E。相当于复制向量
```

```
            给向量的元素赋值：  B(1)=2.0;          -给 B 向量的第 1 维元素赋值为 2.0
            提取向量元素的值：  f=B(2);            -将 B 向量的第 2 维元素提取出来，赋给变量 f

              得到向量的维数：   B.nsize();

*******************************************************************************
                  【注意】:
                  1.本类库的矩阵、向量运算采用值传递
                  2.本类库的矩阵和向量的第一个元素的起始下标为 1，不是 0。这点尤其要注意。这
和 C 语言中的数组不一样
   ****************************************************************************/
   CVector V[3];

   bool test()
   {
    V[0].resize (4);
    V[0](1)=1;
    double k=V[0](1);
    cout<<k;
    return true;
   }

   int main()
   {
    double a[]={1,2,3,4,5,6,7,8,9};
    double b[]={1.12,2.45,5.67,-2.2,3.4,6.4,0,7.1};

    //创建一个 2 行 3 列的零阵 mA
    CMatrix mA(2,3);
    mA(1)=2;
    mA(2)=4;
    mA(3)=mA(2);
    mA(4)=5;////mA(4)其实就是 mA(2,1)
    mA(2,3)=9;
    cout<<"测试 1：\n"<<mA;
    cout<<"mA(4)="<<mA(4)<<"\n";
    cout<<"mA(2,1)="<<mA(2,1)<<"\n";


    //由 mA 来创建一个一模一样的矩阵 mB
    CMatrix mB(mA);
    cout<<"测试 2：\n"<<mB;
```

```
cout<<"A 的内存地址: "<<&mA;
cout<<"\nB 的内存地址: "<<&mB<<"\n";


//resize 矩阵 mA。保留原来 mA 中的前 2×2 个元素
mA.resize(2,2);
cout<<"测试 3: \n"<<mA;
cout<<"A 的内存地址: "<<&mA<<"\n";//可见 resize 后, A 的地址不变


//由数组 a 来创建一个矩阵 mC
CMatrix mC(3,2,a);
cout<<"测试 4: \n"<<mC;


//求 mD 转置
CMatrix mD(2,3,a);
cout<<"测试 5: \n"<<mD<<"\n 转置前 mD 的地址: "<<&mD<<"\n";
mD=transpose(mD);
cout<<mD<<"\n 转置后 mD 的地址: "<<&mD<<"\n";
//可见, 虽然转置后行数、列数互换, 这里仍然可以直接把转置后的矩阵赋给自身
//位于等号左边的矩阵可以自动调整行数和列数。这很便利!
//求逆矩阵
CMatrix mE(2,2,b);
cout<<"测试 6: \n"<<mE;
mE=inv(mE);
cout<<mE;
cout<<inv(mE);//再求逆回来。注意这句不会改变 mE 本身, 但可以显示出再次求逆后的结果
cout<<mE;

//矩阵的连续运算
CMatrix A(2,2,a);
CMatrix B(3,2,a);
CMatrix C(2,3,a);
CMatrix D;
D=inv(A)*transpose(B)+C;//求 D=A 的逆乘以 B 的转置, 再加上 C
//可见, 该类库来进行矩阵的一系列运算非常简单!
cout<<"测试 7: \nA 的逆乘以 B 的转置, 再加上 C\n"<<A<<B<<C<<D;
cout<<A<<inv(A)<<A*inv(A);
cout<<diagprod(A)<<"\n";

////////////////////////////////////
//以下进入向量部分。利用 tvec.h 库
CVector vA(3,a);
```

```
cout<<"测试 8: 向量\n"<<vA;
//由向量构造矩阵
CMatrix mAA(3,1,vA);
cout<<"由向量构造矩阵\n"<<mAA;
//这里，将列向量 vA（3 行、1 列）转化为 3 行、1 列的矩阵，于是就可以使用矩阵计算公式来兼容向量了！
//
//最后调用 diagbyarray 函数来由一个数组来构造一个矩阵。这里矩阵对角线的元素为该数组的元素
CMatrix mBB;
double aa[]={1,2,3,4,5};
diagbyarray(mBB,aa,5);
cout<<"最后一个测试: \n"<<mBB;
diagbyarray(mBB,aa,3);
cout<<"\n"<<mBB;

CVector v1,v2;
v1.resize (4);
v1(1)=2;
v2.resize (4);
cout<<v1;
cout<<v2;
double kkk=v1(1);
cout<<"kkk: "<<kkk<<"\n";

return 0;
}
```

（4）BP 网络的优化及改进

如何对设计出来的 BP 网络进行优化、对算法进行改进，以提高识别率、改进训练时间。

BP 网络存在局部极小值问题，它表现为当训练进行到误差曲面的局部极小值的时候，会陷入在里面而无法跳出来。另外还有个问题就是当隐层神经元数目增多时网络训练时间急速增长的问题。当然，最核心的问题还是提高网络的效率，提高识别率，提高稳定性。

BP 网络优化通常有下面几个方法：多次选择不同的初始权值以找到可以达到最佳识别率的权值；采用加入动量因子的方法；采用可变速率学习（VLBP）；采用模拟退火法；其他各种数值优化算法等等。这里主要介绍、使用两种较为可行的方法。

① 多次选择不同的初始权值。多次选择不同的初始权值，这样就有可能避开局部极小值以达到真正的全局最小值。当选用不同的初始值的时候，网络的训练时间及误差曲线上可以看到较大改进。

② 采用动量法。即在权值增量中加入一个动量因子。这种方法的产生是基于对误差曲线振荡性的观察：如果能平滑轨迹中的振荡将能提高收敛性能，可以用一个低通滤波器来实现。

先来看一个简单的平滑效果的例子：

$$y(k) = y(k-1) + (1-\alpha) \cdot w(k)$$

式中，$w(k)$是滤波器输入，α是动量系数（又称动量因子），$y(k)$是滤波器输出。

可以在图 11-33 中简单明了的看出轨迹被平滑了。

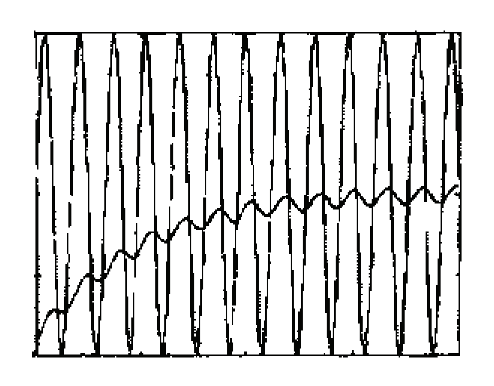


图 11-33　动量的平滑效应

基于以上现象，提出了用于修正 BP 算法的动量方法。

在原始的公式中，有：

$$\Delta W^m(k) = -\alpha\ S^m\ (a^{m-1})^T$$

$$\Delta b^m(k) = -\alpha\ S^m$$

在加入动量因子γ后，每次的权值修正改为：

$$\Delta W^m(k) = \gamma \cdot \Delta W^m(k-1) - (1-\gamma) \cdot \alpha \cdot S^m \cdot (a^{m-1})T$$

$$\Delta b^m(k) = \gamma \cdot \Delta b^m(k-1) - (1-\gamma) \cdot \alpha \cdot S^m$$

在经过以上综合方法的优化后，可以相当明显的改进训练误差曲线、提高识别率、降低训练时间。

本文没有提供各种 BP 网络优化的源代码，只提供了一种思路。读者可以自行探索。通过训练时间的缩短和识别率的提高可以看出效果。

11.5.3　程序总体编程框架

在讲述完以上程序的各个实现细节（图像读取、图像预处理、神经网络识别）之后，笔者将其连起来，综合讲述一下该程序的设计思路及流程。

首先，需要设置一些全局变量以方便参数的保存和传递。全局变量定义在 mydiblib.h 中。如下所列举：

```
//标准归一化的宽度默认为 8
int w_sample=8;

//标准归一化的高度默认为 16
int h_sample=16;

//文件是否加载的标志
bool fileloaded;
```

```
//是否已经输入了归一化高、宽信息的标志
bool gyhinfoinput;

//是否标准归一化完毕的标志
bool gyhfinished;

//记录当前图片中分割出来的数字字符的个数
int digicount;

//去噪函数中调用的临时全局变量
int m_lianXuShu;

//定义两个 Crect 型的链表，一个用于保存与读写，另一个用来作备份
CRectLink m_charRectCopy;
CRectLink m_charRect;

//定义两个 HDIB 型的链表。用来存放分割、归一化之后的图像字符块
HDIBLink  m_dibRect;
HDIBLink  m_dibRectCopy;

//当前系统中图像的公有句柄
HDIB m_hDIB;

//打开文件的文件名
CString strPathName;

//保存文件的文件名
CString strPathNameSave;
```

现在，来贯穿一遍程序完整的流程，以给读者一个完整清晰的认识：

（1）打开含字符的图像文件（训练的时候为训练样本图片，识别的时候为含有未知数字的图片），并将图像文件中除文件头部分的其他所有信息读入内存也即句柄 m_hDIB。

（2）打开输入归一化宽度和高度的对话框，接收用户输入。这里推荐用户使用宽度为 8 高度为 16 的归一化指标。此归一化指标只由用户输入一次。

（3）然后进行图像的预处理。预处理的步骤依次如下：灰度化→二值化→锐化→去离散噪声→整体倾斜调整→字符分割→尺寸标准归一化→紧缩重排。至此，原先散落在图像中的亮度不一、大小不一、斜度不一、粗细不一、同时还含有噪声的数字已经被提取出来，亮度一致，尺寸标准化，去掉了倾斜，并且在很大程度上也去除了噪声。

（4）下面就进入 BP 网络训练部分。训练之前首先要输入 BP 训练的参数，这里主要是训练步长和允许误差。BP 训练的时候，首先要对步骤（3）的结果来进行特征提取，然后将提取的特征送入 BP 网络进行训练。如果网络设计得当，参数选择也正确的话，那么 BP 网络将会在有限步内收敛。训练完毕的 BP 网络将网络的权值保存到文件中，以便下一步识别的时候可以直接调用。

（5）现在可以真正进入识别部分了。打开一幅含有数字的图像，然后是预处理，然后是特征提取，最后将提取后的特征送入 BP 网络，通过网络的输出可以判定输入的字符，以实现数字字符的识别。识别的结果显示在屏幕上，同时也存储在文件中以保存。

11.5.4　程序使用说明、测试及注意事项

下面简要的讲述一下该程序的运行，并对该 BP 网络设计的效果作以评测。

程序主界面如图 11-34 所示。



图 11-34　程序主界面

程序主要功能都通过这几个菜单来实现。常用的菜单也已经集成到了工具栏上，像打开、保存、重新加载、训练网络、识别功能都已经做成了工具栏上的按钮。

“文件”菜单主要负责文件的打开、保存以及重新加载（取消一切更改，重新打开）。“图像预处理”菜单主要来对图像进行预处理，含归一化信息的输入。“神经网络识别”菜单主要实现对网络的训练以及识别。这 3 个菜单的执行效果如下：

“文件”菜单如图 11-35 所示，“图像预处理”菜单如图 11-36 所示。



图 11-35　“文件”菜单

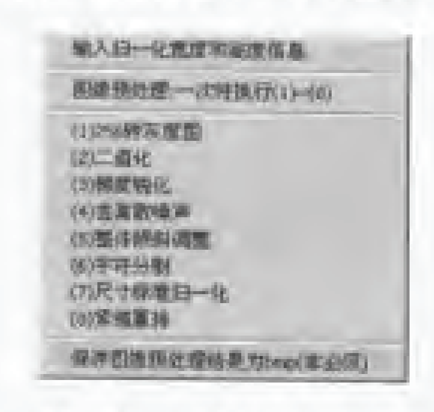


图 11-36　“图像预处理”菜单

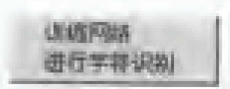


图 11-37　“神经网络识别”菜单

下面来打开一幅训练样本进行训练，效果如图 11-38。

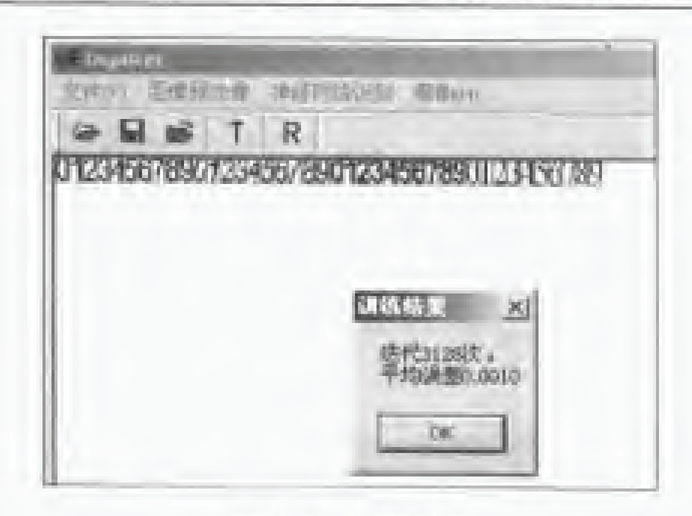


图 11-38 训练结果

然后，打开一幅测试图片来进行识别，如图 11-39 所示。

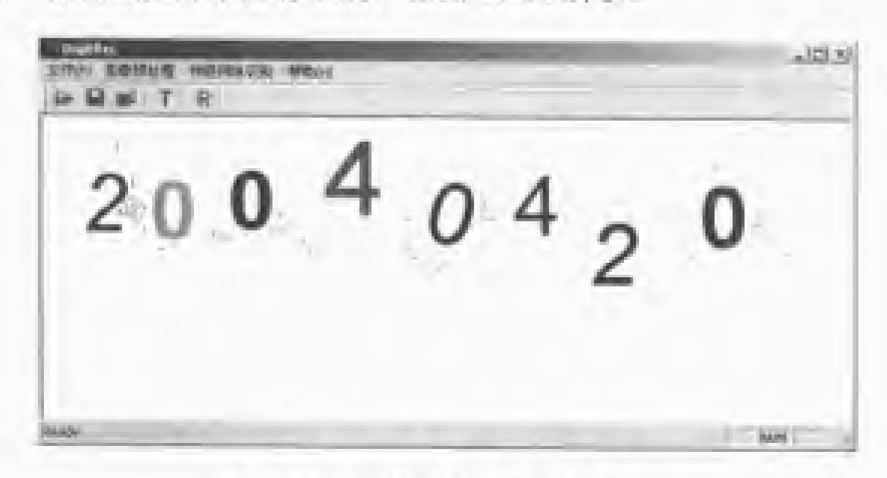


图 11-39 测试图片（待识别图片）

进行图像预处理的结果（归一化后图片过小，可能无法清晰的显示了），如图 11-40 所示。



图 11-40 预处理后的测试图片

下面就可以点击工具栏 T R 中的“R”按钮来进行识别了。识别结果如 11-41 所示。保存到文件中的结果如图 11-42 所示。

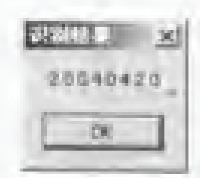


图 11-41 对测试图片的识别结果

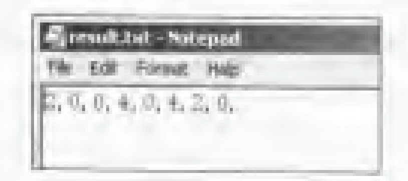


图 11-42 保存在文本文件中的识别结果

至此，程序已经完整地运行了一次由训练到识别的过程。再下一次进行识别的时候，由于本次训练好的网络已经保存了下来，那么就可以直接识别了，无需再训练。当然如果识别的对象发生了较大变化而难以识别的时候，就需要重新制作训练样本并进行识别了。

这是对另一幅图片的识别，如图 11-42 所示。

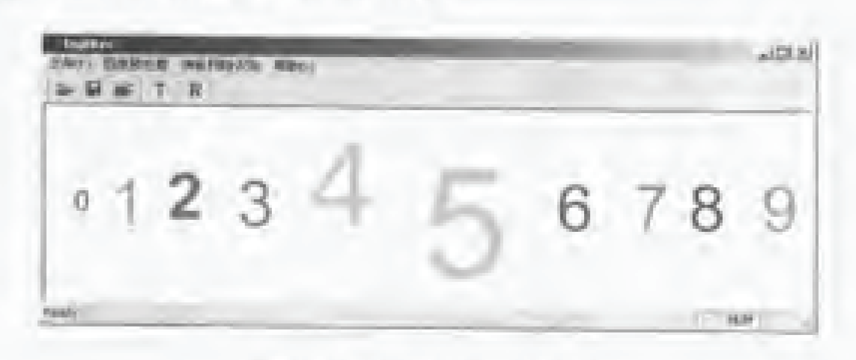


图 11-43 另一幅待识别图片

识别结果如图 11-44 所示。



图 11-44 识别结果

最后来说一下使用该程序的注意事项。

（1）该程序设计时考虑了数字字符的很多变化情况，具备良好的适应性，但识别率对于倾斜字符或者不同字体的字符来说就不是很高了，甚至出现无法识别的情况。如 11-45 图所示，字符“3”倾斜后就无法识别出来了。

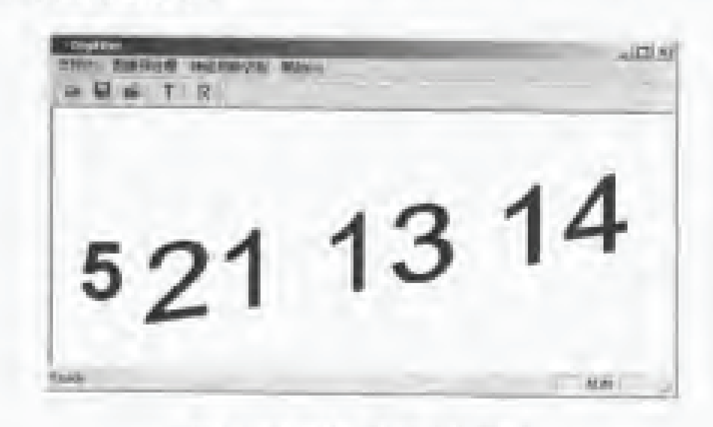


图 11-45 另一幅待识别图片

识别结果如图 11-46 所示。



图 11-46 相应的识别结果

此时读者可以通过加大训练样本的数目来解决。将训练样本数目增大到 150 个，并考虑进来各种倾斜角度的以及其他几种常用字体，那么该 BP 网络将会有更加普遍的适应性和更高的识别率。

（2）考虑程序设计的方便，该程序中设计 BP 网络的目标输出的时候，假定了目标输出为 0，1，2，3，4，5，6，7，8，9 的循环。所以读者在自行设计训练样本的时候一定要注意，不要使用"0，2，1，3"这样的训练样本，这会导致无法训练成功。应该使用"0，1，2，3"这样的训练样本。一定要遵循由 0 到 9 的顺序来设计训练样本。对于待识别图片当然没有这种要求。

（3）在执行图像预处理时，尽量使用"一次性图像预处理"。若要单独执行每一步，请务必记住：每一步要按顺序执行，且只能执行一次。否则将会有意想不到的后果。比如说，就像梯度锐化，连续梯度锐化 3 次，那图片上的数字基本上就没了。这也就无法完成下一步的分割等操作了。

（4）识别时不要再改动归一化信息。否则无法识别。归一化信息应该在训练之前设定，而且仅仅设定一次。识别时归一化信息与训练时候的一致，特征提取才能进行，识别才能进行。

（5）待识别的图片要与训练好的权值（win.dat 和 whi.dat）位于同一目录中。

11.6 本章小结

本章详细分析了一个数字识别系统，介绍了许多图像图形处理的方法，以及在处理字符识别中有针对性的算法。并介绍了利用神经网络的相关知识以及用神经网络进行数字识别的整个过程。

在该 BP 网络的设计及使用过程中主要存在下面几个难题：无法从理论上严格的确定最佳的层数和隐层的神经元数目，这只能通过不断测试、对比来求最佳；存在局部极小值问题，以导致网络没有收敛到最小误差便停止训练了，这里解决的核心办法就是多次选择不同的初始权值并在平坦区域加快学习速度，还有动量因子方法，以使其尽量可以跳出这个局部极小值；训练时间过长的问题；误差曲线振荡的问题，可以采用低通滤波器来改善；对于加噪过多的数据识别率不高的问题，可以通过优化网络结构来改进等。这些问题都在本文中或多或少的描述和解决。借此，笔者提出这些问题，希望可以引起读者的注意。有兴趣的读者可以在本程序的基础上自行开发更为优化的 BP 网络。

本章所用到的图像处理和神经网络知识，读者可以参考 Martin T.Hagan 的《Neural Network Design》和冈萨雷斯的《数字图像处理》。

第 12 章 车牌定位系统

12.1 系统简介

车牌定位系统是进行车牌自动识别中的一个重要部分，只有正确地获取整个图像中的车牌部分，才能正确的对车牌进行文字识别。本章处理后的图像，利用第 11 章介绍的程序，就能够进行车牌文字的识别。

12.2 系统基本技术要求

下面是系统具体要达到的基本技术要求：

（1）每张图片的处理时间不能大于 0.5s；

（2）车牌的获取准确率要达到 100%；

（4）对图片噪声具有较强的适应性；

（5）系统要能长时间无故障运行；

（6）系统的操作要求简单。

12.3 系统中用到的关键技术

在本系统中用到了大量图像处理中的相关技术，其中重要的处理过程如灰度化、预处理、二值化、削弱背景干扰、用自定义模板中值滤波、牌照搜索以及区域裁剪等。

12.4 系统软硬件平台

12.4.1 系统的硬件平台

因为在系统运行的过程当中，主要进行的都是图像处理，在这个过程当中要进行大量的数据处理，所以处理器和内存要求比较高，CPU 要求主频在 600Hz 以上（含 600Hz），内存在 128MB 以上（含 128MB）。

12.4.2 系统的软件平台

系统可以运行于 Windows 98、Windows 2000 或者 Windows XP 操作系统下。程序调试时，需要使用 Microsoft Visual C++ 6.0（SP6）。

12.5 系统实现

系统在实现的过程中，先分解成两个大块，就是图像预处理模块和图像裁剪。具体的实现过程请参照下面的流程图。

12.5.1 系统流程图

车牌定位系统的处理流程图如图 12-1 所示。

图像输入

↓

图像预处理

↓

二值化
（用初始阈值 T 对图像 A 进行二值化得到二值化图像 B）

↓

削弱背景干扰
（对图像 B 作简单的相邻像素灰度值相减，得到新的图像 G）

↓

用自定义模板中值滤波，区域灰度基本被赋值为0
（考虑到文字是由许多短竖线组成，而背景噪声有一大部分是孤立噪声，用模板 (1,1,1,1,1)T 对 G 进行中值滤波得到除掉了大部分干扰的图像 C）

↓

牌照搜索
利用垂直投影法检测车牌垂直位置
利用水平投影法检测车牌水平位置

↓

区域裁剪，截取车牌子图像

图 12-1　车牌定位系统处理流程

12.5.2　程序实现

根据前面的处理流程分析，接下来将通过 Visual C++来实现该车牌定位系统。由于基础的理论知识在前面已经得到了很多的分析和说明，本章重点是介绍代码实现。

程序运行界面如图 12-2 所示。



图 12-2　车牌定位系统运行界面

1．24 位真彩色图转换成 256 级灰度图

为了能够方便的进行后期的数字图像处理，需要将 24 位真彩色图转化成 256 色的灰度图后进行处理。

在图 12-2 中，单击“转”按钮即实现图像的灰度转换，具体代码如下。

```
//按钮"转的代码处理事件"
void CTypeRecView::OnFILE24ToGray()
{
CTypeRecDoc* pDoc = GetDocument(); //获取文档
LPSTR lpDIB;                       //指向 DIB 的指针

lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) pDoc->GetHDIB());

//调用 ConvertToGrayScale 函数实现灰度
ConvertToGrayScale(lpDIB);
::GlobalUnlock((HGLOBAL) pDoc->GetHDIB());
pDoc->SetModifiedFlag(TRUE);
pDoc->UpdateAllViews(NULL);
}
```

```
/**************************************************************************
 *
 * 函数名称:
 *   ConvertToGrayScale()
 *
 * 参数:
 *   LPSTR lpDIB          - 指向源 DIB 图像指针
 *
 * 返回值:
 *   BOOL                      - 成功返回 TRUE,否则返回 FALSE。
 *
 * 说明:
 *   该函数将 24 位真彩色图转换成 256 级灰度图
 *
 **************************************************************************/
BOOL WINAPI ConvertToGrayScale(LPSTR lpDIB)
{

 LPSTR lpDIBBits;                    //指向 DIB 的象素的指针
 LPSTR lpNewDIBBits;                 //指向 DIB 灰度图图像(新图像)开始处象素的指针
    LONG lLineBytes;
 unsigned char * lpSrc;              //指向原图像象素点的指针
 unsigned char * lpdest;             //指向目标图像象素点的指针

    unsigned  char *ired,*igreen,*iblue;

    long lWidth;                     //图像宽度和高度
 long lHeight;

 long i,j;          //循环变量

 //DIB 宽度
 lWidth = ::DIBWidth(lpDIB);

 //DIB 高度
 lHeight = ::DIBHeight(lpDIB);
 RGBQUAD  *lpRGBquad;

 //INFOHEADER 后为调色板
 lpRGBquad = (RGBQUAD *)&lpDIB[sizeof(BITMAPINFOHEADER)];
    if(::DIBNumColors(lpDIB) == 256)  //256 色位图不作任何处理
 {
     return TRUE;
    }
```

```
if(::DIBNumColors(lpDIB) != 256)  //非 256 色位图将它灰度化
{
    lLineBytes = WIDTHBYTES(lWidth*8*3);
    lpdest= new BYTE[lHeight*lWidth];
    lpDIBBits = (LPSTR)lpDIB + sizeof(BITMAPINFOHEADER);//指向 DIB 象素
    for(i = 0;i < lHeight; i++)
        for(j = 0;j < lWidth*3; j+=3)
        {
            ired   = (unsigned char*)lpDIBBits + lLineBytes * i + j + 2;
            igreen = (unsigned char*)lpDIBBits + lLineBytes * i + j + 1;
            iblue  = (unsigned char*)lpDIBBits + lLineBytes * i + j ;
            lpdest[i*lWidth + j/3]  = (unsigned  char)((*ired)*0.299 +
(*igreen)*0.588 + (*iblue)*0.114);
        }

    //需要做 3 件事情: 1、修改 INFOHEADER 2、增加调色板 3、修改原图像灰度值
    LPBITMAPINFOHEADER  lpBI;
    lpBI  =  (LPBITMAPINFOHEADER)lpDIB;
    lpBI->biBitCount = 8;

    //设置灰度调色板
    for(i = 0;i < 256;i++)
    {
      lpRGBquad[i].rgbRed   = (unsigned char)i;
      lpRGBquad[i].rgbGreen = (unsigned char)i;
      lpRGBquad[i].rgbBlue  = (unsigned char)i;
      lpRGBquad[i].rgbReserved = 0;
    }

    lpNewDIBBits= ::FindDIBBits(lpDIB);  //找到 DIB 图像象素起始位置
    lLineBytes=WIDTHBYTES(lWidth * 8);
    //修改灰度值
    for(i = 0;i < lHeight; i++)
      for(j = 0;j < lWidth; j++)
      {
          lpSrc = (unsigned char*)lpNewDIBBits + lLineBytes * i+ j ;
          *lpSrc=lpdest[i*lWidth+j];
      }
    delete lpdest;
}
return true;
}
```

2. 二值化

用一初始阈值 T 对图像 A 进行二值化得到二值化图像 B，初始阈值 T 的确定方法是：选择阈值 $T=G_{max}-(G_{max}-G_{min})/3$，$G_{max}$ 和 G_{min} 分别是最高、最低灰度值。

该阈值对不同牌照有一定的适应性，能够保证背景基本被置为 0，以突出牌照区域。

```
//按钮“1”的单击处理代码
void CTypeRecView::OnTest11()
{
 CTypeRecDoc* pDoc=GetDocument();   //获得文档
 LPSTR lpDIB;                       //指向 DIB 的指针
 LPSTR lpDIBBits;                    //指向 DIB 的象素的指针

    LONG lLineBytes;
 unsigned char * lpSrc;              //指向原图像象素点的指针

    long lWidth;                      //图像宽度和高度
 long lHeight;

 long i,j;          //循环变量

 //复制图像
 OnEditCopy();

 lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) pDoc->GetHDIB());//获得当前位图
 // 找到 DIB 图像象素起始位置
 lpDIBBits = ::FindDIBBits(lpDIB);
 lWidth = ::DIBWidth(lpDIB);   //DIB 宽度
 lHeight = ::DIBHeight(lpDIB); //DIB 高度

 // 计算图像每行的字节数
 lLineBytes = WIDTHBYTES(lWidth * 8);

 long lCount[256];
 for(i=0;i<256;i++)
 {
     lCount[i]=0;  //清零
 }
 if(::DIBNumColors(lpDIB) != 256)  //256 色位图不作任何处理
 {
     return;
 }
 for(i=0;i<lHeight;i++)
 {
     for(j=0;j<lWidth;j++)
     {
```

```
            lpSrc=(unsigned char *)lpDIB+lLineBytes*i+j;
            lCount[*(lpSrc)]++;
        }
    }

//求窗口变换的上限和下限
//
long temp[16];
int k=0;

for(k=0;k<16;k++)
{
    temp[k]=0;
    for(i=k*16;i<(k+1)*16;i++)
        temp[k]+=lCount[i];
}

long max=0;
int t=0;
for(k=15;k>=0;k--)
{
    if(temp[k]>max)
    {
        max=temp[k];
        t=k;
    }
}

int bLow=0,bUp=0;
bLow=(t-1)*16;
//  bUp=(t+5)*16;

//  bLow=100;
bUp=255;
// 阈值
INT bThre;

bThre=(INT)((2*bUp+bLow)/3);

// 更改光标形状
BeginWaitCursor();

// 调用 ThresholdTrans()函数进行阈值变换
ThresholdTrans(lpDIBBits, ::DIBWidth(lpDIB), ::DIBHeight(lpDIB), bThre);
```

```
pDoc->SetModifiedFlag(TRUE);
pDoc->UpdateAllViews(NULL);
::GlobalUnlock((HGLOBAL) pDoc->GetHDIB());
// 恢复光标
EndWaitCursor();
}

/*************************************************************************
 *
 * 函数名称:
 *   ThresholdTrans()
 *
 * 参数:
 *   LPSTR lpDIBBits    - 指向源 DIB 图像指针
 *   LONG  lWidth       - 源图像宽度 (象素数)
 *   LONG  lHeight      - 源图像高度 (象素数)
 *   BYTE  bThre        - 阈值
 *
 *  返回值:
 *  BOOL                   - 成功返回 TRUE, 否则返回 FALSE。
 *
 * 说明:
 * 该函数用来对图像进行阈值变换。对于灰度值小于阈值的象素直接设置
 * 灰度值为 0; 灰度值大于阈值的象素直接设置为 255。
 *
 ************************************************************************/
BOOL WINAPI ThresholdTrans(LPSTR lpDIBBits, LONG lWidth, LONG lHeight, BYTE
bThre)
{
 // 指向源图像的指针
 unsigned char*  lpSrc;

 // 循环变量
 LONG    i;
 LONG    j;

 // 图像每行的字节数
 LONG    lLineBytes;

 // 计算图像每行的字节数
 lLineBytes = WIDTHBYTES(lWidth * 8);

 // 每行
```

```
for(i = 0; i < lHeight; i++)
{
    // 每列
    for(j = 0; j < lWidth; j++)
    {
        // 指向 DIB 第 i 行, 第 j 个象素的指针
    lpSrc = (unsigned char*)lpDIBBits + lLineBytes * (lHeight - 1 - i) + j;
        // 判断是否小于阈值
        if ((*lpSrc) < bThre)
        {
            // 直接赋值为 0
            *lpSrc = 0;
        }
        else
        {
            // 直接赋值为 255
            *lpSrc = 255;
        }
    }
}

// 返回
return TRUE;
}

//复制当前图像，该函数在 CTypeRecView::OnTest11()中被调用
void CTypeRecView::OnEditCopy()
{
// 复制当前图像

// 获取文档
CTypeRecDoc* pDoc = GetDocument();

// 打开剪贴板
if (OpenClipboard())
{
    // 更改光标形状
    BeginWaitCursor();

    // 清空剪贴板
    EmptyClipboard();

    // 复制当前图像到剪贴板
    SetClipboardData (CF_DIB, CopyHandle((HANDLE) pDoc->GetHDIB()) );
```

```
        // 关闭剪贴板
        CloseClipboard();

        // 恢复光标
        EndWaitCursor();
    }
}
```

3. 削弱背景干扰

对图像 B 作简单的相邻像素灰度值相减，得到新的图像 G，左边缘直接赋值，不会影响整体效果。

```
//按钮“2”单击处理代码
void CTypeRecView::OnTest12()
{
 CTypeRecDoc* pDoc=GetDocument();    //获得文档
 LPSTR lpDIB;                        //指向 DIB 的指针

 lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) pDoc->GetHDIB());//获得当前位图

 //用自定义的模板消弱背景干扰
 myTemplate(lpDIB);

 pDoc->SetModifiedFlag(TRUE);
 pDoc->UpdateAllViews(NULL);
 ::GlobalUnlock((HGLOBAL) pDoc->GetHDIB());
}

/*************************************************************************
 *
 * 函数名称:
 *   Template()
 *
 * 参数:
 *   LPSTR lpDIBBits     - 指向源 DIB 图像指针
 *   LONG  lWidth        - 源图像宽度（象素数）
 *   LONG  lHeight       - 源图像高度（象素数）
 *   int   iTempH        - 模板的高度
 *   int   iTempW        - 模板的宽度
 *   int   iTempMX          - 模板的中心元素 X 坐标 ( < iTempW - 1)
 *   int   iTempMY          - 模板的中心元素 Y 坐标 ( < iTempH - 1)
 *    FLOAT * fpArray       - 指向模板数组的指针
 *    FLOAT fCoef           - 模板系数
 *
 * 返回值:
```

```
 *   BOOL               - 成功返回 TRUE，否则返回 FALSE。
 *
 * 说明：
 *   该函数用指定的模板（任意大小）来对图像进行操作，参数 iTempH 指定模板
 * 的高度，参数 iTempW 指定模板的宽度，参数 iTempMX 和 iTempMY 指定模板的中心
 * 元素坐标，参数 fpArray 指定模板元素，fCoef 指定系数。
 *
 ***************************************************************************/

BOOL WINAPI myTemplate(LPSTR lpDIB)
{
 // 指向复制图像的指针
 LPSTR   lpNewDIBBits;
 HLOCAL  hNewDIBBits;

 // 指向要复制区域的指针
 unsigned char*  lpDst;

 LPSTR lpDIBBits;                   //指向 DIB 的像素的指针

    LONG lLineBytes;                  // 图像每行的字节数
 unsigned char * lpSrc;             //指向原图像像素点的指针

    long lWidth;                     //图像宽度和高度
 long lHeight;

 // 计算结果
 INT fResult;

 // 找到 DIB 图像像素起始位置
 lpDIBBits = ::FindDIBBits(lpDIB);
 lWidth = ::DIBWidth(lpDIB);   //DIB 宽度
 lHeight = ::DIBHeight(lpDIB); //DIB 高度

 // 计算图像每行的字节数
 lLineBytes = WIDTHBYTES(lWidth * 8);

 // 暂时分配内存，以保存新图像
 hNewDIBBits = LocalAlloc(LHND, lLineBytes * lHeight);

 // 判断是否内存分配失败
 if (hNewDIBBits == NULL)
 {
     // 分配内存失败
```

```
    return false;
}

// 锁定内存
lpNewDIBBits = (char * )LocalLock(hNewDIBBits);

// 初始化图像为原始图像
memcpy(lpNewDIBBits, lpDIBBits, lLineBytes * lHeight);

long i,j;           //循环变量

for(i = 0; i < lHeight-1; i++)
{
    // 每列
    for(j = 0; j < lWidth-1; j++)
    {
        // 指向新 DIB 第 i 行，第 j 个象素的指针
        lpDst=(unsigned char*)lpNewDIBBits + lLineBytes * (lHeight -1-i) + j;
        // 指向 DIB 第 i 行，第 j 个象素的指针
        lpSrc=(unsigned char*)lpDIBBits + lLineBytes * (lHeight - 1 - i) + j;

        fResult=(*lpSrc)-(*(lpSrc+1));

        // 取绝对值
        if(fResult<0) fResult=-fResult;

        // 判断是否超过 255
        if(fResult > 255)
        {
            // 直接赋值为 255
            * lpDst = 255;
        }
        else
        {
            // 赋值
            * lpDst = (unsigned char) (fResult + 0.5);
        }
    }
}

// 复制变换后的图像
memcpy(lpDIBBits, lpNewDIBBits, lLineBytes * lHeight);

// 释放内存
```

```
    LocalUnlock(hNewDIBBits);
    LocalFree(hNewDIBBits);

    // 返回
    return TRUE;
}
```

4. 用自定义模板中值滤波

区域灰度基本被赋值为 0。考虑到文字是由许多短竖线组成，而背景噪声有一大部分是孤立噪声，用模板(1,1,1,1,1)*T* 对 *G* 进行中值滤波，得到除掉了大部分干扰的图像 *C*。

```
//按钮 "3" 的单击处理代码
void CTypeRecView::OnTest13()
{
    // 中值滤波

    // 获取文档
    CTypeRecDoc* pDoc = GetDocument();

    // 指向 DIB 的指针
    LPSTR    lpDIB;

    // 指向 DIB 像素指针
    LPSTR    lpDIBBits;

    // 滤波器的高度
    int iFilterH;

    // 滤波器的宽度
    int iFilterW;

    // 中心元素的 X 坐标
    int iFilterMX;

    // 中心元素的 Y 坐标
    int iFilterMY;

    // 锁定 DIB
    lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) pDoc->GetHDIB());

    // 找到 DIB 图像像素起始位置
    lpDIBBits = ::FindDIBBits(lpDIB);

    // 判断是否是 8-bpp 位图（这里为了方便，只处理 8-bpp 位图的中值滤波，其它的可以类推）
    if (::DIBNumColors(lpDIB) != 256)
    {
```

```
        // 提示用户
        MessageBox("目前只支持 256 色位图的中值滤波! ", "系统提示" ,
            MB_ICONINFORMATION | MB_OK);

        // 解除锁定
        ::GlobalUnlock((HGLOBAL) pDoc->GetHDIB());

        // 返回
        return;
    }

    // 创建对话框

    // 初始化变量值
    iFilterH = 5;
    iFilterW = 1;
    iFilterMX = 0;
    iFilterMY = 2;

    // 更改光标形状
    BeginWaitCursor();

    // 调用 MedianFilter()函数中值滤波
    if (myMedianFilter(lpDIBBits, ::DIBWidth(lpDIB), ::DIBHeight(lpDIB),
        iFilterH, iFilterW, iFilterMX, iFilterMY))
    {

        // 设置脏标记
        pDoc->SetModifiedFlag(TRUE);

        // 更新视图
        pDoc->UpdateAllViews(NULL);
    }
    else
    {
        // 提示用户
        MessageBox("分配内存失败! ", "系统提示" , MB_ICONINFORMATION | MB_OK);
    }

    // 解除锁定
    ::GlobalUnlock((HGLOBAL) pDoc->GetHDIB());

    // 恢复光标
    EndWaitCursor();
```

```
}

/*************************************************************************
 *
 * 函数名称:
 *   MedianFilter()
 *
 * 参数:
 *   LPSTR lpDIBBits      - 指向源 DIB 图像指针
 *   LONG  lWidth         - 源图像宽度(象素数)
 *   LONG  lHeight            - 源图像高度(象素数)
 *   int   iFilterH           - 滤波器的高度
 *   int   iFilterW           - 滤波器的宽度
 *   int   iFilterMX      - 滤波器的中心元素 X 坐标
 *   int   iFilterMY      - 滤波器的中心元素 Y 坐标
 *
 * 返回值:
 *   BOOL                     - 成功返回 TRUE,否则返回 FALSE。
 *
 * 说明:
 *   该函数对 DIB 图像进行中值滤波。
 *
 ************************************************************************/

BOOL WINAPI myMedianFilter(LPSTR lpDIBBits, LONG lWidth, LONG lHeight,
                  int iFilterH, int iFilterW,
                  int iFilterMX, int iFilterMY)
{

 // 指向源图像的指针
 unsigned char*  lpSrc;

 // 指向要复制区域的指针
 unsigned char*  lpDst;

 // 指向复制图像的指针
 LPSTR           lpNewDIBBits;
 HLOCAL          hNewDIBBits;

 // 指向滤波器数组的指针
 unsigned char   * aValue;
 HLOCAL          hArray;

 // 循环变量
```

```
LONG            i;
LONG            j;
LONG            k;
LONG            l;

// 图像每行的字节数
LONG            lLineBytes;

// 计算图像每行的字节数
lLineBytes = WIDTHBYTES(lWidth * 8);

// 暂时分配内存，以保存新图像
hNewDIBBits = LocalAlloc(LHND, lLineBytes * lHeight);

// 判断是否内存分配失败
if (hNewDIBBits == NULL)
{
    // 分配内存失败
    return FALSE;
}

// 锁定内存
lpNewDIBBits = (char * )LocalLock(hNewDIBBits);

// 初始化图像为原始图像
memcpy(lpNewDIBBits, lpDIBBits, lLineBytes * lHeight);

// 暂时分配内存，以保存滤波器数组
hArray = LocalAlloc(LHND, iFilterH * iFilterW);

// 判断是否内存分配失败
if (hArray == NULL)
{
    // 释放内存
    LocalUnlock(hNewDIBBits);
    LocalFree(hNewDIBBits);

    // 分配内存失败
    return FALSE;
}

// 锁定内存
aValue = (unsigned char * )LocalLock(hArray);
```

```
    // 开始中值滤波
    // 行(除去边缘几行)
    for(i = iFilterMY; i < lHeight - iFilterH + iFilterMY + 1; i++)
    {
        // 列(除去边缘几列)
        for(j = iFilterMX; j < lWidth - iFilterW + iFilterMX + 1; j++)
        {
            // 指向新 DIB 第 i 行, 第 j 个象素的指针
            lpDst=(unsigned char*)lpNewDIBBits + lLineBytes *(lHeight-1-i)+ j;

            // 读取滤波器数组
            for (k = 0; k < iFilterH; k++)
            {
                for (l = 0; l < iFilterW; l++)
                {
                //指向 DIB 第 i - iFilterMY + k 行, 第 j - iFilterMX + l 个象素的指针
                    lpSrc=(unsigned char*)lpDIBBits + lLineBytes * (lHeight - 1 -
i + iFilterMY - k) + j - iFilterMX + l;

                    // 保存象素值
                    aValue[k * iFilterW + l] = *lpSrc;
                }
            }

            // 获取中值
            * lpDst = myGetMedianNum(aValue, iFilterH * iFilterW);
        }
    }

    // 复制变换后的图像
    memcpy(lpDIBBits, lpNewDIBBits, lLineBytes * lHeight);

    // 释放内存
    LocalUnlock(hNewDIBBits);
    LocalFree(hNewDIBBits);
    LocalUnlock(hArray);
    LocalFree(hArray);

    // 返回
    return TRUE;
    }


    /***************************************************************************
```

```
*
* 函数名称:
*   GetMedianNum()
*
* 参数:
*   unsigned char * bpArray  - 指向要获取中值的数组指针
*   int   iFilterLen         - 数组长度
*
* 返回值:
*   unsigned char    - 返回指定数组的中值。
*
* 说明:
*   该函数用冒泡法对一维数组进行排序，并返回数组元素的中值。
*
************************************************************************/

unsigned char WINAPI myGetMedianNum(unsigned char * bArray, int iFilterLen)
{
 // 循环变量
 int     i;
 int     j;

 // 中间变量
 unsigned char bTemp;

 // 用冒泡法对数组进行排序
 for (j = 0; j < iFilterLen - 1; j ++)
 {
     for (i = 0; i < iFilterLen - j - 1; i ++)
     {
         if (bArray[i] > bArray[i + 1])
         {
             // 互换
             bTemp = bArray[i];
             bArray[i] = bArray[i + 1];
             bArray[i + 1] = bTemp;
         }
     }
 }

 // 计算中值
 if ((iFilterLen & 1) > 0)
 {
     // 数组有奇数个元素，返回中间一个元素
```

```
        bTemp = bArray[(iFilterLen + 1) / 2];
    }
    else
    {
        // 数组有偶数个元素，返回中间两个元素平均值
        bTemp = (bArray[iFilterLen / 2] + bArray[iFilterLen / 2 + 1]) / 2;
    }

    // 返回中值
    return bTemp;
}
```

5. 牌照搜索

利用水平投影法检测车牌水平位置，利用垂直投影法检测车牌垂直位置。

```
//按钮“4”的单击处理代码
void CTypeRecView::OnTest145()
{
CTypeRecDoc* pDoc=GetDocument();   //获得文档
LPSTR lpDIB;                       //指向 DIB 的指针
   long lWidth;                       //图像宽度和高度
long lHeight;

lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) pDoc->GetHDIB());//获得当前位图
lWidth = ::DIBWidth(lpDIB);   //DIB 宽度
lHeight = ::DIBHeight(lpDIB); //DIB 高度

//水平投影，求取车牌子图像的上下边缘位置
//
myHprojectDIB(lpDIB, lWidth, lHeight,&m_ipzTop, &m_ipzBottom) ;
m_ipzLeft=0;
m_ipzRight=lWidth;

//对含车牌图像进行剪裁，得到车牌高度，原图像宽度的图像
//  OnTempSubrect();

HDIB hDIB;
HDIB hNewDIB;
hDIB=pDoc->GetHDIB();

  //假定的剪裁区域(车牌附近)
//
CRect rect(m_ipzLeft,m_ipzTop,m_ipzRight,m_ipzBottom);
hNewDIB= myCropDIB(hDIB,rect);

// 判断是否剪裁成功
```

```
if (hNewDIB != NULL)
{
    // 获取文档
    CTypeRecDoc* pDoc = GetDocument();

    // 替换DIB，同时释放旧DIB对象
    pDoc->ReplaceHDIB(hNewDIB);

    // 更新DIB大小和调色板
    pDoc->InitDIBData();

    // 设置脏标记
    pDoc->SetModifiedFlag(TRUE);

    // 实现新的调色板
    OnDoRealize((WPARAM)m_hWnd,0);

}

lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) pDoc->GetHDIB());//获得当前位图
lWidth = ::DIBWidth(lpDIB);   //DIB 宽度
lHeight =::DIBHeight(lpDIB); //DIB 高度

//垂直投影
myVprojectDIB(lpDIB, lWidth, lHeight,&m_ipzLeft, &m_ipzRight) ;

pDoc->UpdateAllViews(NULL);
::GlobalUnlock((HGLOBAL) pDoc->GetHDIB());
}

/*************************************************************************
 *
 * 函数名称:
 *   myHprojectDIB()
 *
 * 参数:
 *   LPSTR lpDIBBits    - 指向源DIB图像指针
 *   LONG  lWidth       - 源图像宽度（象素数）
 *   LONG  lHeight      - 源图像高度（象素数）
 *   INT*  iTop         - 车牌上边缘
 *   INT*  iBottom      - 车牌下边缘
 * 返回值:
 *   BOOL               - 成功返回TRUE，否则返回FALSE。
 *
```

```
* 说明:
*   该函数用于对含车牌图像进行水平投影运算，求取车牌子图像的上下边缘位置
*
***************************************************************************/
BOOL WINAPI myHprojectDIB(LPSTR lpDIB, LONG lWidth, LONG lHeight,
                  int* iTop, int* iBottom)
{
LPSTR lpDIBBits;                    //指向DIB的象素的指针

   LONG lLineBytes;                   // 图像每行的字节数
unsigned char * lpSrc;             //指向原图像象素点的指针

unsigned char pixel;                 //像素值

int i,j;

// 计算结果
INT* iResult;

INT pzBottom,pzTop;

bool findPZ=false;

// 找到DIB图像象素起始位置
lpDIBBits = ::FindDIBBits(lpDIB);

// 计算图像每行的字节数
lLineBytes = WIDTHBYTES(lWidth * 8);

iResult=new INT[lHeight];

for(i=0;i<lHeight;i++)
    iResult[i]=0;

for(j=lHeight/5;j<lHeight*0.8;j++)
{
    iResult[j]=0;
    for(i=0;i<lWidth;i++)
    {
        lpSrc=(unsigned char*)lpDIBBits+lLineBytes*j+i;
        pixel=(unsigned char)(*lpSrc);
        if(pixel==255)
        {
            iResult[j]++;
```

```
        }
    }
    if((!findPZ)&&iResult[j]>12)
    {
        pzBottom=j;
        findPZ=true;
    }
    if(findPZ&&iResult[j]<12)
    {
        pzTop=j;
        break;
    }
}
pzTop=pzBottom+55;
pzBottom=pzBottom-20;   //微量调整
*iBottom=lHeight-pzBottom;
*iTop=lHeight-pzTop;
return true;
}


/*************************************************************************
*
* 函数名称:
*   myVprojectDIB()
*
* 参数:
*   LPSTR lpDIBBits    - 指向源 DIB 图像指针
*   LONG  lWidth       - 源图像宽度（象素数）
*   LONG  lHeight      - 源图像高度（象素数）
*   INT*  iLeft        - 车牌左边缘
*   INT*  iRight       - 车牌右边缘
* 返回值:
*   BOOL               - 成功返回 TRUE，否则返回 FALSE。
*
* 说明:
*   该函数用于对含车牌图像进行垂直投影运算，求取车牌子图像的左右边缘位置
*
*************************************************************************/
BOOL WINAPI myVprojectDIB(LPSTR lpDIB, LONG lWidth, LONG lHeight,
                          int* iLeft, int* iRight)
{
 LPSTR lpDIBBits;                    //指向 DIB 的象素的指针
```

```
LONG lLineBytes;                    // 图像每行的字节数
unsigned char * lpSrc;             //指向原图像象素点的指针

unsigned char pixel;                //像素值

int i,j;

// 计算结果
INT* iResult;

INT pzLeft,pzRight;

bool findPZ=false;

// 找到DIB图像象素起始位置
lpDIBBits = ::FindDIBBits(lpDIB);

// 计算图像每行的字节数
lLineBytes = WIDTHBYTES(lWidth * 8);

iResult=new INT[lWidth];

for(i=0;i<lWidth;i++)
    iResult[i]=0;

for(i=lWidth/5;i<lWidth*0.8;i++)
{
    iResult[i]=0;
    for(j=0;j<lHeight;j++)
    {
        lpSrc=(unsigned char*)lpDIBBits+lLineBytes*j+i;
        pixel=(unsigned char)(*lpSrc);
        if(pixel==255)
        {
            iResult[i]++;
        }
    }
    if((!findPZ)&&iResult[i]>10)
    {
        pzLeft=i;
        findPZ=true;
    }
    pzRight=pzLeft+150;
}
```

```
*iLeft=pzLeft-10;
*iRight=pzRight+10;
return true;
}
```

6. 截取车牌子图像

经过前面的若干步骤后，下面要进行精确的车牌区域的提取，单击“5”按钮完成截取车牌子图像的功能。

```
//按钮“5”单击处理代码
void CTypeRecView::OnTest16()
{
OnEditPaste();

 //调用 OnTempSubrect 函数
 OnTempSubrect();
 OnEditPaste();
}

//剪裁指定区域图像
void CTypeRecView::OnTempSubrect()
{

 CTypeRecDoc* pDoc = GetDocument();
 HDIB hDIB,hNewDIB;
 hDIB=pDoc->GetHDIB();

 long lWidth;                    //图像宽度和高度
 long lHeight;

 // 指向 DIB 的指针
 LPSTR   lpDIB;

 lpDIB = (LPSTR) ::GlobalLock((HGLOBAL) pDoc->GetHDIB());//获得当前位图
 // 找到 DIB 图像象素起始位置
 lWidth = ::DIBWidth(lpDIB);   //DIB 宽度
 lHeight = ::DIBHeight(lpDIB); //DIB 高度

    //假定的剪裁区域(车牌附近)
 CRect rect(m_ipzLeft,m_ipzTop,m_ipzRight,m_ipzBottom);
 hNewDIB= myCropDIB(hDIB,rect);

 if (OpenClipboard())
 {
     EmptyClipboard();
```

```
        SetClipboardData (CF_DIB, CopyHandle((HANDLE) hNewDIB ));
        CloseClipboard();
    }
}

/*************************************************************************
 * 函数名称：
 *   myCropDIB()
 *
 * 参数：
 *   HDIB hDIB              - 指向源 DIB 图像句柄
 *   LPRECT lpRect           - 指向要剪裁的矩形区域
 *
 * 返回值：
 *   HDIB                   - 返回裁剪后的矩形区域图像句柄。
 *
 * 说明：
 * 该函数用于对指定图像进行指定区域裁剪操作。
 *
 * 要求目标图像为 255 个灰度值的灰度图像。
 *************************************************************************/

HDIB WINAPI myCropDIB(HDIB hDIB, LPRECT lpRect)
{

 LPBITMAPINFO lpbmi = NULL;
    LPBYTE       lpSourceBits, lpTargetBits, lpResult;
    HDC          hDC = NULL, hSourceDC, hTargetDC;
    HBITMAP  hSourceBitmap, hTargetBitmap, hOldTargetBitmap, hOldSourceBitmap;
    DWORD dwSourceBitsSize, dwTargetBitsSize, dwTargetHeaderSize;
 int          nWidth, nHeight;
 HDIB         hNewDIB;
 DWORD        dwSize;

 // Get DIB pointer
 if (! hDIB)
 {
     return NULL;
 }
 LPBITMAPINFO lpSrcDIB = (LPBITMAPINFO)GlobalLock(hDIB);
 if (! lpSrcDIB)
 {
     return NULL;
 }
```

```
    dwTargetHeaderSize = sizeof( BITMAPINFOHEADER ) + PaletteSize((LPSTR)
lpSrcDIB);
    lpbmi = (LPBITMAPINFO)malloc( dwTargetHeaderSize );
  memcpy(lpbmi, lpSrcDIB, dwTargetHeaderSize);
  nWidth = RECTWIDTH(lpRect);
  nHeight = RECTHEIGHT(lpRect);
    lpbmi->bmiHeader.biWidth = nWidth;
    lpbmi->bmiHeader.biHeight = nHeight;

  hDC = ::GetDC( NULL );
    hTargetBitmap = CreateDIBSection( hDC, lpbmi, DIB_RGB_COLORS, (VOID
**)&lpTargetBits, NULL, 0 );
    hSourceBitmap = CreateDIBSection( hDC, lpSrcDIB, DIB_RGB_COLORS, (VOID
**)&lpSourceBits, NULL, 0 );
    hSourceDC = CreateCompatibleDC( hDC );
    hTargetDC = CreateCompatibleDC( hDC );

    dwSourceBitsSize = lpSrcDIB->bmiHeader.biHeight * BytesPerLine((LPBYTE) &
(lpSrcDIB->bmiHeader));
    dwTargetBitsSize = lpbmi->bmiHeader.biHeight * BytesPerLine((LPBYTE) &
(lpbmi->bmiHeader));
    memcpy( lpSourceBits, FindDIBBits((LPSTR)lpSrcDIB), dwSourceBitsSize );
    lpbmi->bmiHeader.biSizeImage = dwTargetBitsSize;

    hOldSourceBitmap = (HBITMAP)SelectObject( hSourceDC, hSourceBitmap );
    hOldTargetBitmap = (HBITMAP)SelectObject( hTargetDC, hTargetBitmap );

  // 将老的 BMP 写入一个新的 BMP
    BitBlt(hTargetDC, 0, 0, nWidth, nHeight, hSourceDC, lpRect->left,
lpRect->top, SRCCOPY);

    // 清空然后删除 DCs
    SelectObject( hSourceDC, hOldSourceBitmap );
    SelectObject( hTargetDC, hOldTargetBitmap );
    DeleteDC( hSourceDC );
    DeleteDC( hTargetDC );
    ::ReleaseDC( NULL, hDC );

    GdiFlush();

    // 申请足够空间
  dwSize = dwTargetHeaderSize + dwTargetBitsSize;
  hNewDIB =(HDIB)GlobalAlloc(GHND, dwSize);
```

```
    lpResult  = (LPBYTE)GlobalLock(hNewDIB);//malloc( dwTargetHeaderSize +
dwTargetBitsSize );
    memcpy( lpResult, lpbmi, dwTargetHeaderSize );
    memcpy( FindDIBBits( (LPSTR)lpResult ), lpTargetBits, dwTargetBitsSize );

    // 最后清空
    DeleteObject( hTargetBitmap );
    DeleteObject( hSourceBitmap );
    free( lpbmi );
  GlobalUnlock(hDIB);
  GlobalUnlock(hNewDIB);

    return hNewDIB;
}

//粘贴图像
void CTypeRecView::OnEditPaste()
{
 // 粘贴图像

 // 创建新 DIB
 HDIB hNewDIB = NULL;

 // 打开剪贴板
 if (OpenClipboard())
 {
     // 更改光标形状
     BeginWaitCursor();

     // 读取剪贴板中的图像
     hNewDIB = (HDIB) CopyHandle(::GetClipboardData(CF_DIB));

     // 关闭剪贴板
     CloseClipboard();

     // 判断是否读取成功
     if (hNewDIB != NULL)
     {
         // 获取文档
         CTypeRecDoc* pDoc = GetDocument();

         // 替换 DIB，同时释放旧 DIB 对象
         pDoc->ReplaceHDIB(hNewDIB);
```

```
        // 更新 DIB 大小和调色板
        pDoc->InitDIBData();

        // 设置脏标记
        pDoc->SetModifiedFlag(TRUE);

        // 实现新的调色板
        OnDoRealize((WPARAM)m_hWnd,0);

        // 更新视图
        pDoc->UpdateAllViews(NULL);
    }
    // 恢复光标
    EndWaitCursor();
}
}

//实现新的调色板
LRESULT CTypeRecView::OnDoRealize(WPARAM wParam, LPARAM)
{
 ASSERT(wParam != NULL);

 // 获取文档
 CTypeRecDoc* pDoc = GetDocument();

 // 判断 DIB 是否为空
 if (pDoc->GetHDIB() == NULL)
 {
     // 直接返回
     return 0L;
 }

 // 获取 Palette
 CPalette* pPal = pDoc->GetDocPalette();
 if (pPal != NULL)
 {
     // 获取 MainFrame
     CMainFrame* pAppFrame = (CMainFrame*) AfxGetApp()->m_pMainWnd;
     ASSERT_KINDOF(CMainFrame, pAppFrame);

     CClientDC appDC(pAppFrame);

     CPalette* oldPalette=appDC.SelectPalette(pPal, ((HWND)wParam) != m_hWnd);
```

```
        if (oldPalette != NULL)
        {
            UINT nColorsChanged = appDC.RealizePalette();
            if (nColorsChanged > 0)
                pDoc->UpdateAllViews(NULL);
            appDC.SelectPalette(oldPalette, TRUE);
        }
        else
        {
        TRACE0("\tCCh1_1View::OnPaletteChanged中调用SelectPalette()失败！\n");
        }
    }

    return 0L;
}
```

12.5.3 程序效果测试

前面介绍了车牌定位系统的核心代码，下面通过两副图像测试本系统的定位效果。首先打开第一幅图（如图12-2所示）。

单击工具栏中的"转"按钮，效果如图12-3所示。



图12-3　单击"转"按钮后的效果

注意：由于书本打印的时候，也是以灰度打印的，因此书本中看图12-3和图12-2的效果差不多，但是读者可以运行程序观察效果。

单击"1"按钮，效果如图12-4所示。

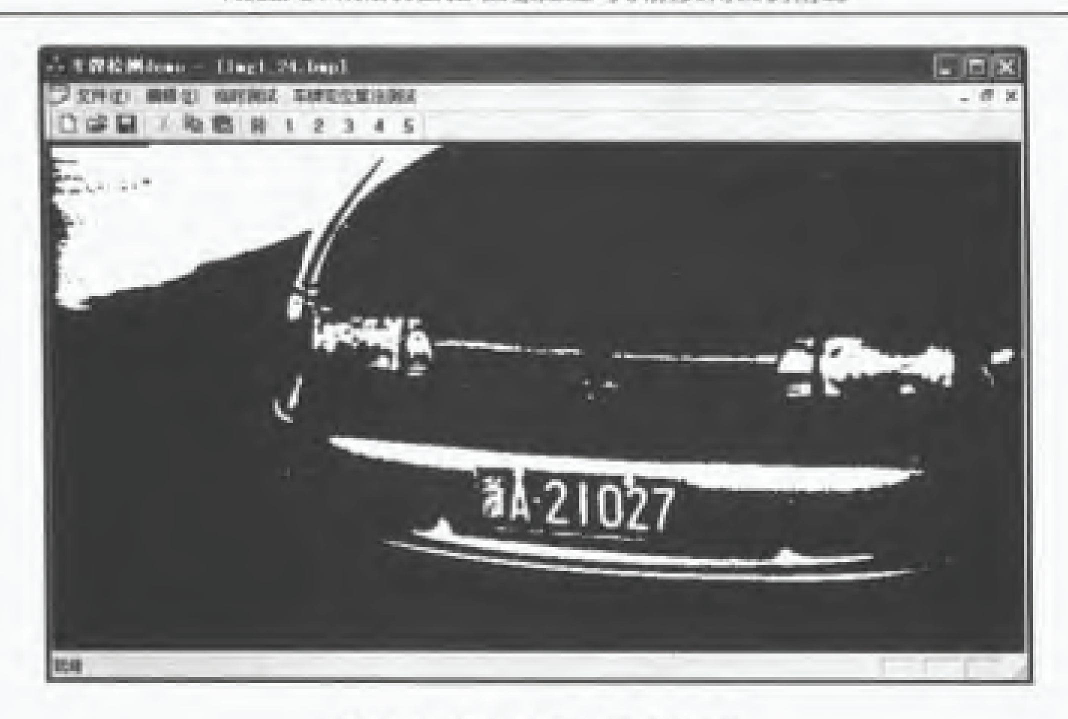


图 12-4　单击“1”按钮后的效果

单击“2”按钮，效果如图 12-5 所示。

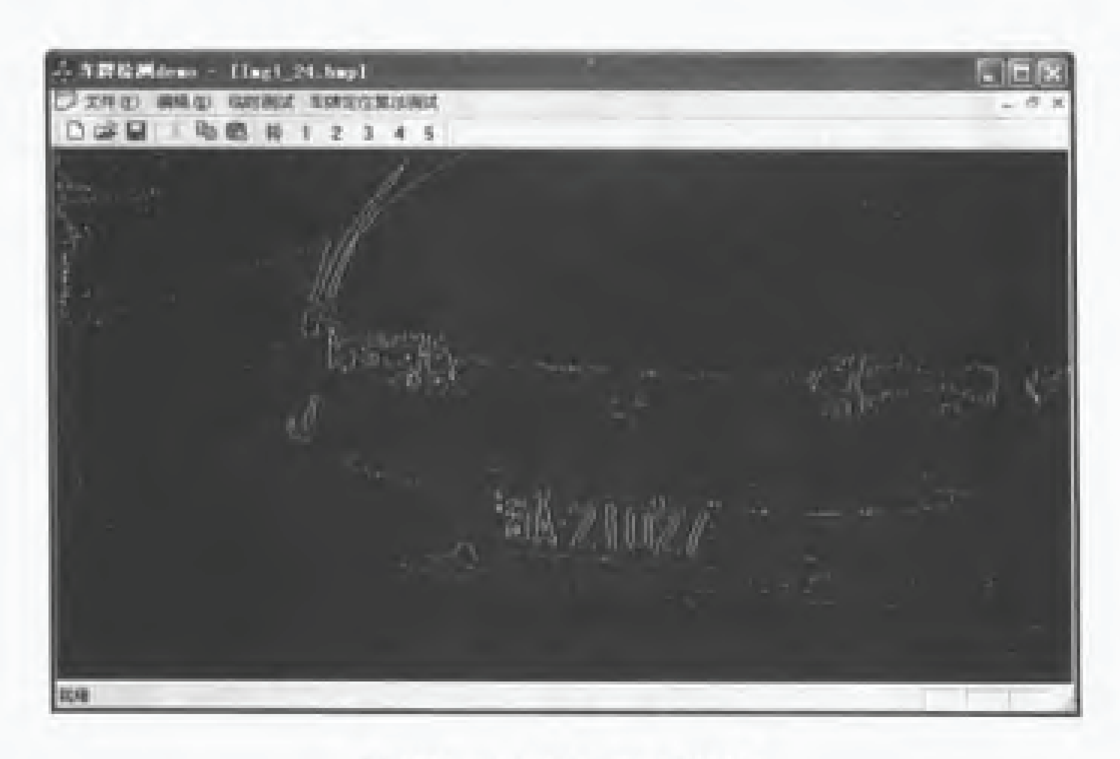


图 12-5　单击“2”按钮后的效果

单击“3”按钮，效果如图12-6所示。

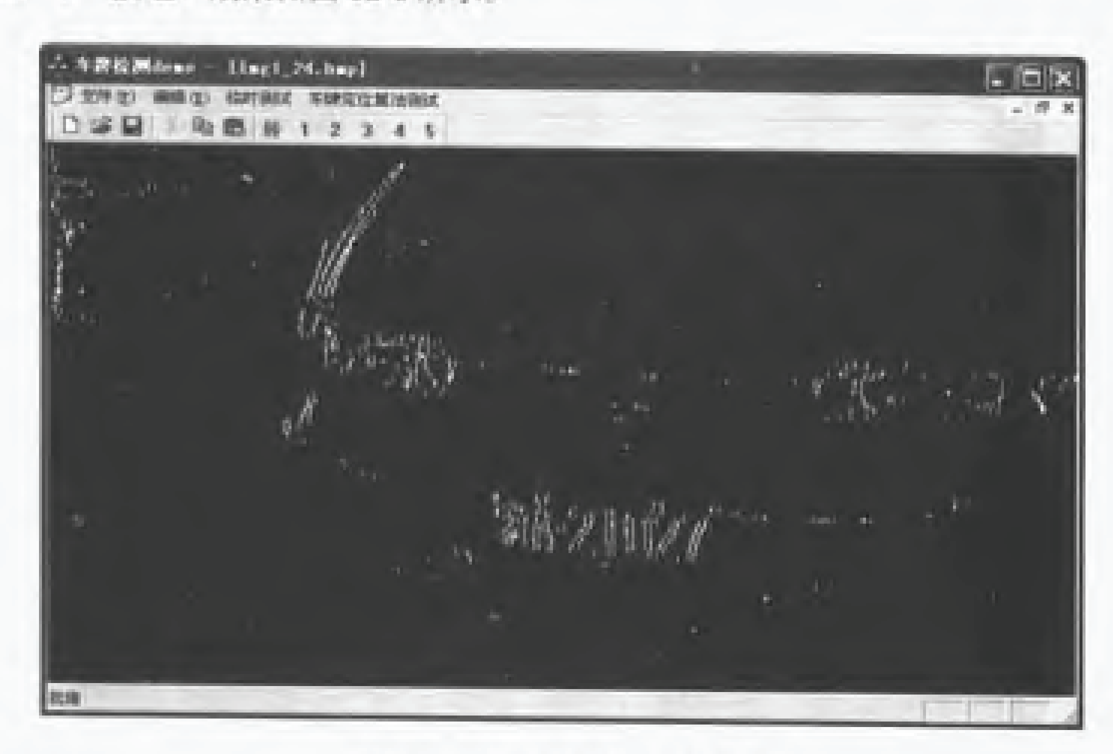


图12-6　单击“3”按钮后的效果

单击“4”按钮，效果如图12-7所示。

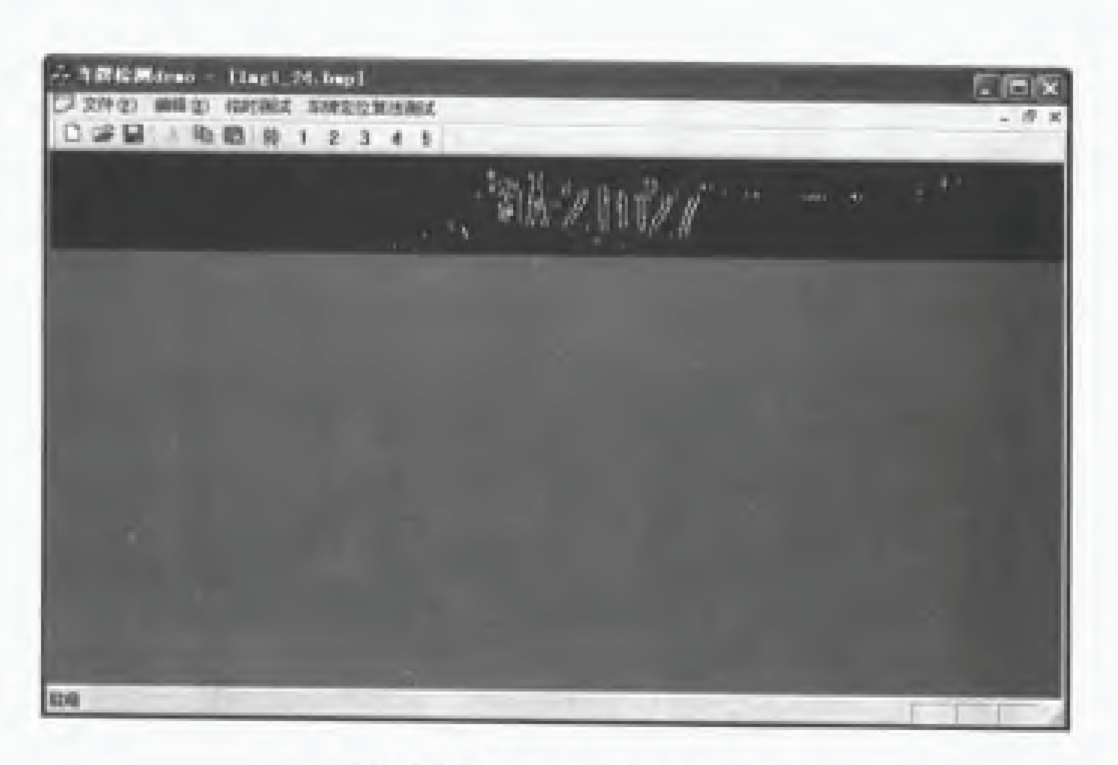


图12-7　单击“4”按钮后的效果

单击“5”按钮，效果如图 12-8 所示。



图 12-8 单击“5”按钮后的效果

从图 12-8 可以看出，本系统对车牌进行的定位是比较精确的。
下面再通过另外一副图像进行测试，如图 12-9 所示。



图 12-9 原始图

单击"转"按钮，效果如图 12-10 所示。



图 12-10　单击"转"按钮后的效果

单击"1"按钮，效果如图 12-11 所示。

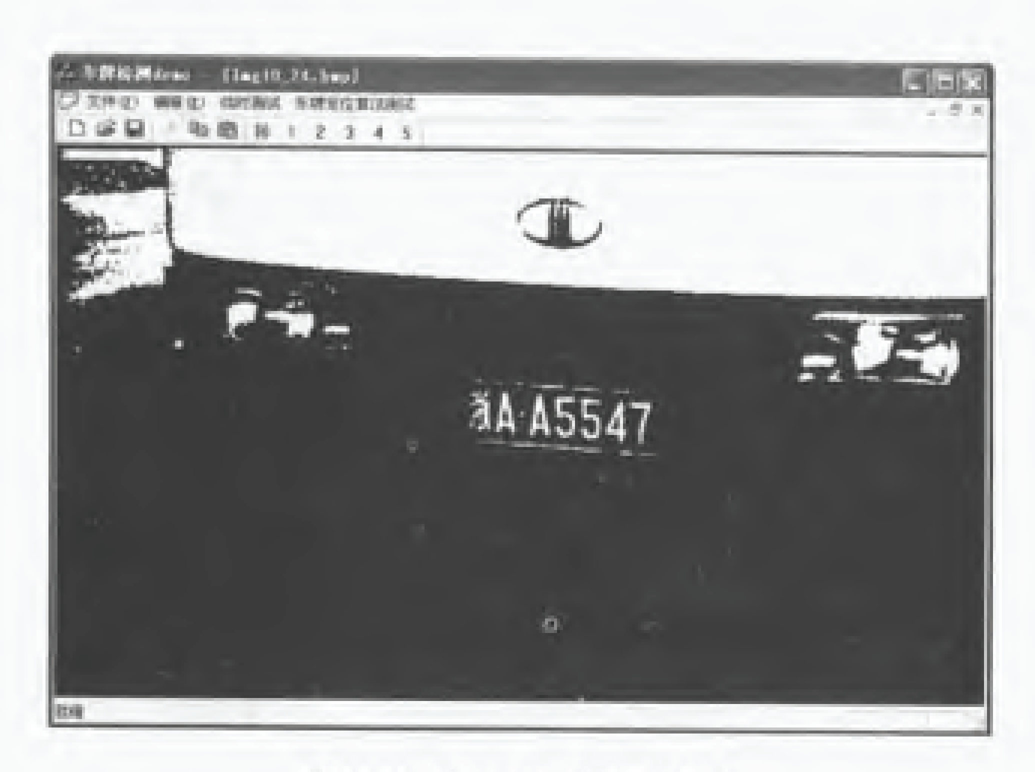


图 12-11　单击"1"按钮后的效果

单击“2”按钮，效果如图 12-12 所示。

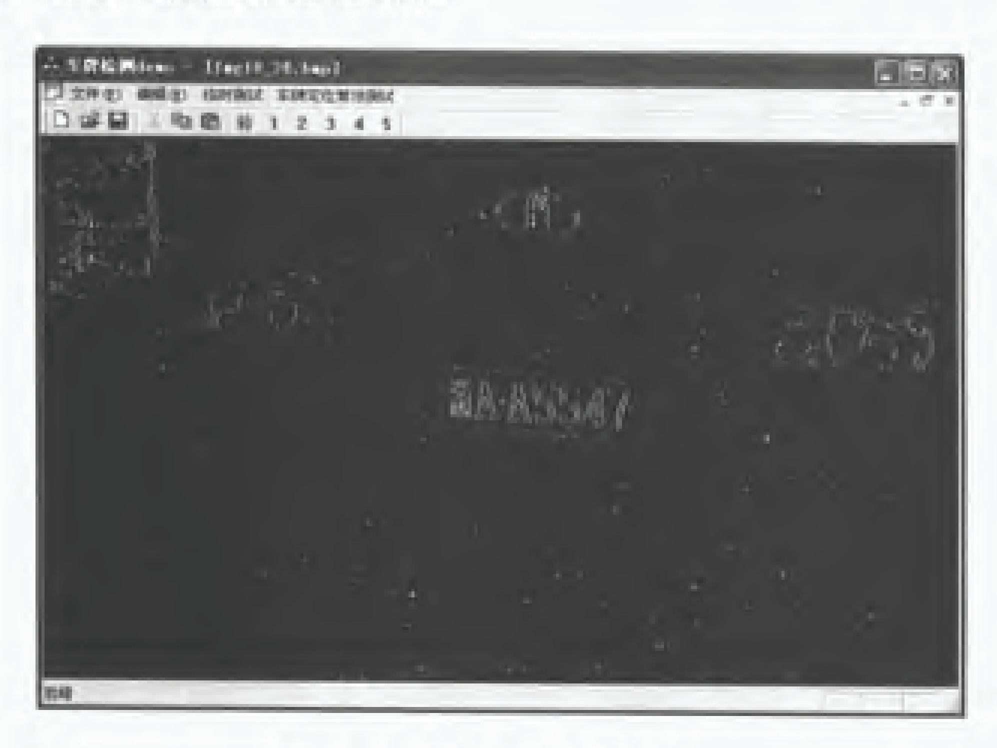


图 12-12　单击“2”按钮后的效果

单击“3”按钮，效果如图 12-13 所示。

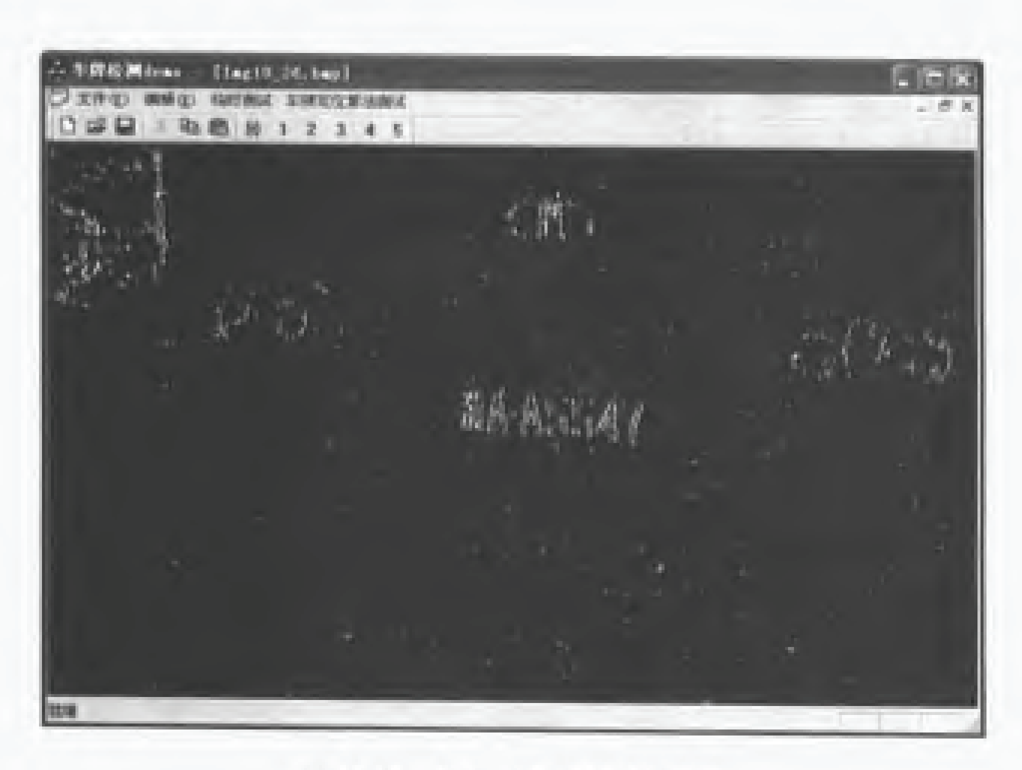


图 12-13　单击“3”按钮后的效果

单击“4”按钮，效果如图 12-14 所示。



图 12-14　单击“4”按钮后的效果

单击“5”按钮，效果如图 12-15 所示。

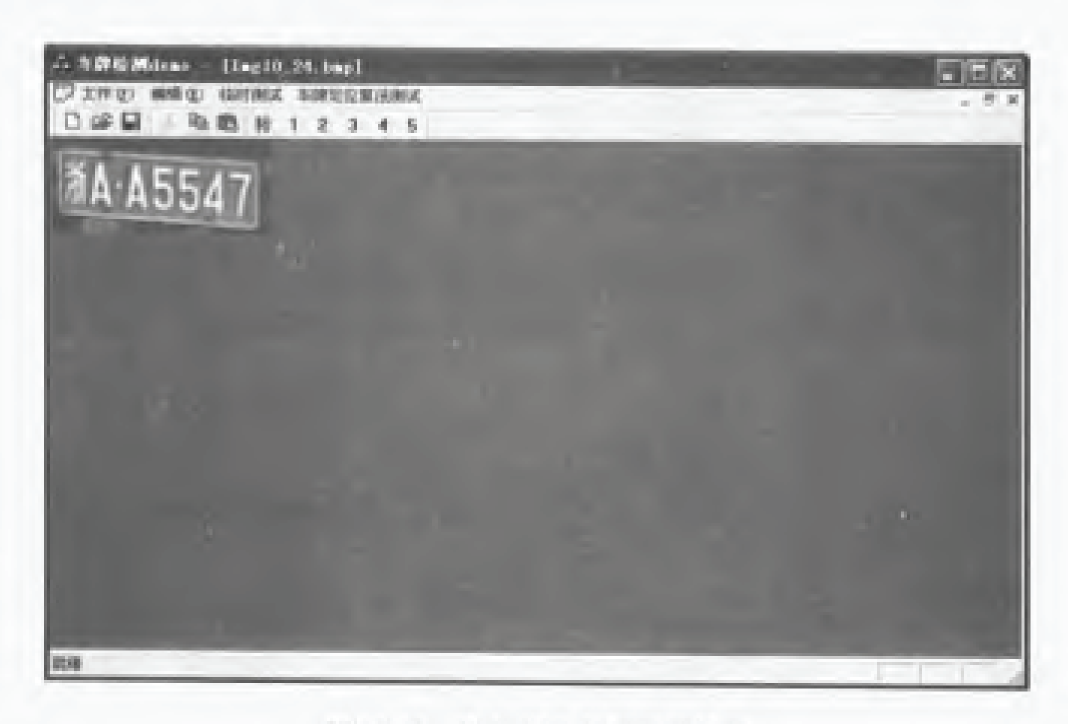


图 12-15　单击“5”按钮后的效果

12.6 本章小结

本章介绍了一个完整车牌识别系统中的一个核心的技术即车牌定位，这是整个车牌识别技术的基础，同时该步骤操作效果的好坏与否直接关系到后一步的识别效果。本程序实现起来不是很困难，关键是要掌握整个处理的流程以及一些参数的设置。

本章部分代码可参阅人民邮电出版社出版的《Visual C++数字图像处理（第二版）》（何斌、马天予等编著）。